DOI:
10.1016/j.infoandorg.2017.04.001

Document Version
Peer reviewed version

GOVERNING OPEN SOURCE SOFTWARE THROUGH COORDINATION PROCESSES

Maha Shaikh
Warwick Business School
The University of Warwick
United Kingdom

Ola Henfridsson
Warwick Business School
The University of Warwick
United Kingdom

Abstract

Governance provides the authoritative framework for coordinating activities in open source development. Prior studies of open source governance have largely focused on its changing nature over time. In this work, we argue that the nature of governance varies across open source communities, and, in its evolution, multiple traces of authority may co-exist. We propose that such multiplicity can be understood by close examination of the authoritative structures embedded in coordination processes. We collected eight years of data on the coordination related to version control of the Linux kernel. Drawing on in-depth qualitative analysis, we investigate how coordination processes with different authoritative structures come together in the governance of open source software. We trace four coordination processes (autocratic clearing, oligarchic recursion, federated self-governance, and meritocratic idea-testing), each grounded in different authoritative structures (autocracy, oligarchy, federation, meritocracy) with their own form of legitimation. We offer a two-fold contribution in this paper. First, we enhance the open source governance literature by advancing a new theoretical perspective in which governance is seen as a configuration of coordination processes. Configurations give complementary support and are a source of tension and renewal. Second, we articulate a view on the conceptual relationship between governance and coordination where these concepts are understood as a duality, both working together to give rise to efficient and dynamic organizing in open source.

INTRODUCTION

Open source development is a powerful means to create and maintain software. The most significant open source communities such as the Linux Kernel successfully gather the contributions of thousands of distributed developers to further evolve its software. To make such coordination possible, version control software is used to track, trace and archive the complete development process. It essentially allows distributed development to take place where the number of developers has grown beyond a small, manageable number. At the same time, it is also imperative for realizing governance of the open source community because access and different roles of responsibility are embedded within its design. In this regard, version control software offers a meeting point for governance and coordination.

Prior open source literature pays attention to both aspects. First, there is a significant body of research dealing with governance (De Noni et al. 2011; Markus 2007; O'Mahony 2007). Governance provides the authoritative framework for organizing open source software development (O'Mahony and Ferraro 2007). As an example, such authority might be reflected in the decision rights over changes in core modules of the software. Second, in addition to this vertical dimension of open source, coordination of the many developers involved is needed for preserving the software's installed base in terms of prior contributions of software code (Fogel 1999; Fogel 2005). It is also vital for effectuating new ones (von Hippel 2001; von Krogh et al. 2005). Yet, despite the significance of vertical authority and horizontal coordination, little has been done to investigate the close relationship between governance and coordination.

In this paper, we propose that governance and coordination can be seen as a duality where both coexist and work in parallel (Farjoun 2010), and both are required simultaneously to be effective. We also propose coordination processes as a concept for understanding how this duality plays out in the course of developing open source software. We refer to coordination processes as the common sets of rules, instructions, and activities that operationalize a specific authoritative structure. In open source software, coordination processes are embedded in the software tools used for version control (Cornford et al. 2010). In addition, they are embedded in the social structures of the open source organization as reflected in communication patterns (Bezroukov 1999; Raymond 1999) and values (Orek and Nov 2008).

Another significant aspect of open source communities with implications for governance relates to its innate emphasis on contributors self-selecting their own tasks. Developers are not tied to a community through a contract so there is no obligation to contribute, instead open source communities rely on different motivations of developers like delayed extrinsic gratification, and a compelling intrinsic need to contribute (Roberts et al. 2006; Spaeth et al. 2015; von Krogh et al. 2012). To cater for the manifold reasons that might underlie developers' commitment, there exist multiple, and sometimes heterogeneous, authoritative structures observed as the governance of open source software evolves (Aaltonen and Lanzara 2015; Markus 2007; Meyer and Montagne 2007; Midha and Bhattacherjee 2012; O'Mahony 2007; O'Mahony and Ferraro 2007; Weber 2005); as one form of governance transforms into another (Bryson et

al. 2014; Daily et al. 2003; Fielding 1999; Garud et al. 2006); and, as more than one form of governance co-exist and eventually converge (O'Mahony and Ferraro 2007). In studies of open source governance, it is therefore imperative to develop conceptions that can cater for the multiplicity of open source governance. We advance the view that in-depth examination of coordination processes and their underlying authoritarian structures is significant for understanding this aspect of open source governance. We define open source governance as a configuration of multiple authoritative structures (each embedded in a coordination process) that guide and steer activities, tasks, motivations and effort towards a collective, and mutually understood goal.

Rather than finding evidence of convergence (cf. O'Mahony and Ferraro 2007) towards a single form of governance, our in-depth study of the Linux Kernel reveals how multiple coordination processes reflecting heterogeneous authoritative structures collectively shaped the governance process. This motivated us to further investigate coordination processes and the governance of open source software by posing the following research question: *how do coordination processes with different authoritative structures come together in the governance of open source software?*

We offer a two-fold contribution in this paper. First, we enhance the open source governance literature by advancing a new theoretical perspective in which governance is seen as configurations of coordination processes, which gives both complementary support and triggers for tension and renewal. Governance is then multiple and not singular in such large, and complex projects and each configuration of coordination processes encapsulates this multiplicity while allowing for complex coordination to occur. Second, we articulate a conception of the relationship between governance and coordination as a duality, where they work together to reinforce each other to give rise to change.

CONCEPTUAL BACKGROUND

Governance is a shared basis of authority (O'Mahony and Ferraro 2007). Rooted in the ideas of Weber (1946) and Etzioni (1959), authority in an organizational setting typically encompasses a mix of legitimacy, ties, and obedience (Harrison 1960). As an example, bureaucracy as an authority structure has

seen significant research (Adler and Borys 1996; Crozier 1964; Kallinikos 2004; Meyer 1968; Ouchi 1980; Riccucci et al. 2016; Weber 1946). Some other examples include adhocracy (Miller 1987; Miller 1990; Mintzberg 1980; Mintzberg and McHugh 1985), alliances (Faems et al. 2008; Gulati 1998; Han et al. 2012; Osborn and Hagedoorn 1997; Ring and van de Ven 1992), and networks (Davis and Eisenhardt 2011; Ingram and Torfason 2010; Peng et al. 2013; Sundararajan et al. 2013).

Perhaps because of its image of representing something profoundly new in terms of organizing (Crowston et al. 2007; Crowston and Scozzi 2002; Dafermos 2001; Gallivan 2001), open source software has attracted its own body of governance studies. However, a closer examination of this literature reveals how governance in open source displays variety in terms of its underlying authority structure. In fact, our literature review helped us distinguish three types of authority reflected in this body of work: centralized, libertarian, and collective (see Table 1).

Table 1: Views of Authority Structures in Open Source Governance

Authority Structures	Definition	Example references
Centralized	Shared understanding that the central core of developers know best how to manage and organize development work.	(Aaltonen and Lanzara 2015; Crowston and Howison 2005; Dahlander and O'Mahony 2011; Koch and Schneider 2002; Kogut and Metiu 2001; Moon and Sproull 2000; Tullio and Staples 2014; Weber 2005)
Libertarian	Shared understanding that individual freedom is very important, and that all members should be able act autonomously and voice their opinion.	(Dahlander and O'Mahony 2011; de Laat 2007; De Noni et al. 2011; De Noni et al. 2013; Fitzgerald 2006; Gallivan 2001; Howison and Crowston 2014; Kuwabara 2000; O'Mahony and Ferraro 2007; Raymond 1999)
Collective	Shared understanding that the needs of the many outweigh those of the few.	(Fielding 1999; Hemetsberger and Reinhardt 2009; Markus 2007; Mockus et al. 2002; O'Mahony and Ferraro 2007; Shah 2006; Sharma et al. 2002)

Centralized authority reflects a shared understanding that the central core of actors, typically key developers, know best how to manage and coordinate development work (Aaltonen and Lanzara 2015; Dahlander and Frederiksen 2012; Dahlander and O'Mahony 2011; Koch and Schneider 2002; Kogut and Metiu 2001; Tullio and Staples 2014; Weber 2005). Peripheral actors are happy to support the core team

unless it is seen to cross radical boundaries of acceptability and ideology. A study of the online user community of Propellerhead in Sweden showcases how so-called "cosmopolitans" bring innovative ideas into the community yet the core developers exercise a deep sense of centralized authority over the community's direction and well-being (Dahlander and O'Mahony 2011). This understanding is not resisted by other community members because their belief is that the central core knows the project best and has worked hard to earn their reputation and leadership roles. This authority structure also reveals the norms of the community that allow less influential actors in the community the choice and ability to work towards becoming more relevant through serious and innovative code contributions over time (Dahlander and O'Mahony 2011).

Second, *libertarian* authority posits that individual level freedom is imperative, and that all members should be able to act autonomously and voice their opinion freely as equals (de Laat 2007; De Noni et al. 2011; De Noni et al. 2013; Feller and Fitzgerald 2002; Fitzgerald 2006; Gallivan 2001; Howison and Crowston 2014; Kuwabara 2000; Raymond 1999). The BibDesk project offers conceptual clarity on how numerous individual developers work to build quality software in a layered approach (Howison and Crowston 2014). Software is built feature by feature by distinct developers with little focus on the larger project but rather what is evident is a micro perspective on the code and work at hand. This individualistic layering approach relies on what has been coded previously rather than attempting grand new designs (Howison and Crowston 2014).

Collective authority, our third and final category, suggests a view where the needs of the many outweigh those of the few, so individual freedom may be curtailed if it is seen to be the right thing to do for the common good (Fielding 1999; Hemetsberger and Reinhardt 2009; Markus 2007; O'Mahony and Ferraro 2007; Sharma et al. 2002). A significant instance of collective authority has been provided by O'Mahony and Ferarro (2007). Their study of the Debian project observes how the governance model in different phases of an open source project is slowly evaluated and evolves to accommodate the changing nature of the community and its needs. The need to evaluate the authority structure in their study was driven by the collective need and recognition that the current governance form was inadequate in some

manner. The belief system did not provide a good match with how work was being governed so the rules of negotiation with management were questioned, evaluated and re-built to reveal a new *collective* order.

Duality of Governance and Coordination

As outlined above, the open source governance literature can be divided into three categories depending on their view on authority: centralized, libertarian, and collective. However, consistent with Etzioni's (1959) point that any organization can only have one seat of authority, existing studies tend to presume a singularity in each governance model. Even in cases where hybrid governance is discussed, the premise of the work is that a new singular authority structure emerges through the process of hybridization (De Noni et al. 2013; Markus 2007; O'Mahony 2007; O'Mahony and Ferraro 2007; Shah 2006; Tullio and Staples 2014). This rich literature also tellingly tends to conflate governance with coordination (Alexy et al. 2013; Dahlander and O'Mahony 2011; Feller et al. 2008). Arguably, similar concerns have been raised previously, especially in the CSCW literature (Malone 1987; Olson et al. 2001; Olson and Olson 2000; Schmidt and Simonee 1996). In CSCW work the artefact, for the most part, is seen as apolitical and simply as a tool of coordination (Malone 1987; Schmidt and Bannon 1992; Schmidt and Simonee 1996). At the same time this work tends to emulate larger, management studies of governance where singular authority is understood to be the norm.

Table 2: Conceptual Constructs

Concept	Definition
Authoritative structure	A coherent arrangement of qualities that provide legitimacy for a particular coordination process.
Coordination process	The common sets of rules, guidelines, and activities that operationalize a specific authoritative structure.
Governance process	A configuration of multiple authoritative structures (each embedded in a coordination process) that guide and steer activities, tasks, motivations and effort towards a collective, and mutually understood goal.

As we will outline in this paper, the link between governance and coordination is important in order to accommodate the dynamically changing governance of a growing and evolving open source community, and allowing multiple forms of coordinating governance processes to co-exist. We draw on the idea of governance and coordination as a duality rather than dualism (Farjoun 2010). A dualism often compares and relates two antithetical, or competing ideas where they work in contradiction to the other (Farjoun 2010). Such ideas are evident in literature and practice on open source governance, yet our study of the Linux Kernel case suggests a duality of governance and coordination. A duality, as opposed to a dualism, draws a relationship between two concepts that are distinct yet in their unfolding over time work together to strengthen, corroborate, and refine each other. Armed with this perspective we craft our idea of coordination processes and provide empirical evidence to explain how these separate yet interdependent processes are entangled and work to reinforce each other rather than only pull in opposition.

The concept of coordination process helps us to establish how governance unfolds in open source development in every day work processes. We define coordination processes *as the common sets of rules, guidelines, and activities that operationalize a specific authoritative structure.* Such processes consist of rules that provide explicit and implicit guidelines for the coordinated activity. For instance, the shared and openly accessible FAQ pages for all projects set out formal guidelines for working on, and contributing to, an open source project. There are often wiki pages that offer input into the learning process of a new open source developer. Also, there are informal guidelines, typically accessible through the ideologies (cf. Barrett et al. 2013) underpinning the authority structure of an open source development project. Such ideologies may be communicated and appreciated in the practice of coordinating in the open source community, and they typically define what are seen as powerful contributions to the software. In this regard, the common sets of rules are sanctioned in a particular understanding of how the community should function (see Table 2).

Returning to our main research question of *how coordination processes with different authoritative structures come together in the governance of open source software*, we embarked on a case study of the Linux Kernel project.

RESEARCH DESIGN AND METHODS

To address the research question related to how coordination processes with different authoritative structures come together in open source governance, we conducted an in-depth case study (Hargadon and Douglas 2001; Kieser 1994; Mason et al. 1997a; Mason et al. 1997b) of version control use and adoption in the development of the Linux Kernel. Version control is software used for coordinating software development. It supports the development process by tracking and archiving revisions to the software over time. Version control software ensures that code contributions will be handled effectively. There are various version control options on the market, and the decision to adopt any one is often based on both technical as well as political reasons because each tool offers different types of governance.

Case Setting and Selection

The Linux Kernel is an ongoing, high-profile open source development project. Its success is rated not simply by the number of developers it is able to attract, the persistence of the project (considering how many open source projects simply die within a few months), but the fact that it has been taken up by so many other communities and companies where different mutations of it have emerged. For instance, Ubuntu, a very successful Debian based Linux operating system has been created and established by the company called Canonical.

There were a number of reasons why we selected version control for the Linux Kernel as our empirical focus. First, the setting offers the complexity needed for studying coordination processes and governance over time. This enables generation of a rich account of dynamics related to coordination processes. Second, it offers valuable and longitudinal data related to the evolution of governance. Linux Kernel development began in 1991. There is public archival access to the entire email communication carried out between the Linux Kernel developers from 1995 onwards to present. Like many, if not all open source projects, the distributed nature of collaborating entails a greater dependence on such coordination tools such as email and version control software. The mailing list in particular is accessible to the public and is a rich and solid source of decisions, discussions and debates held by the community over time. This being one of the earliest cases of open source governance, it offered an abundance of data related to the areas of

coordination tools, management style, as well as governance models for coordinating software contributions of developers. And finally, the Linux kernel case involved a multitude of events when parallel coordination processes marked with different authoritative structures co-existed. The issue of version control software was especially relevant then since the choice of version control to be adopted by the community was contested and offers a useful setting for studying interaction of coordination processes.

Data Collection

The primary data for this research was gathered from the Linux Kernel mailing list archive [LKML]. This site is kept up to date by the University of Indiana and claims to include every email message passed between Linux kernel developers. Other LKML sites were also used to refine searches and points of reference. Each site offered some unique facilities, which proved helpful when crosschecking the completeness of the collected data material.

The data collection covered all the threads and messages related to version control from June 1995 to June 2003. The data was collected in chronological order to reduce the likelihood of missing any relevant threads. Often one thread evolved into another one with a different title, and unless the researcher closely immersed herself into the data material and its sequences of events (Langley 1999), important exchange could have been missed. Each link on the LKML-Indiana website was searched by keywords. Eight years' worth of data yielded a massive amount of information (word count: around 1,200,000), which covered the most critical period of time in terms of version control and coordination. Prior to June 1995, the Linux kernel project was not large enough to require significant version control. After June 2003, the version control software question was less contested. However, to ensure that relevant data was not neglected, the first author examined the list archive on this topic for some five years after.

The LKML website allows the messages on each page to be sorted in three different ways, date, subject, and author-wise. In particular, the *subject* sort was helpful as it broke all the messages down into their respective threads, thus making it possible to download each message related to every identified thread. All the messages were copied and pasted into a text document in readiness for analysis with the help

of content analysis software, Atlas.ti. The first author continued this process for all the messages in the time period of June 1995 – June 2003. The URLs were saved for each email to ensure the ability to return to the original text should the need arise.

The search keywords were regularly adapted as the story unfolded. As the researcher became more acquainted with the story and various threads she was able to adapt the search terms. This necessarily entailed reading a number of threads that potentially could have been of use, but were later discarded for their lack of direct contribution to this study. Over time saturation was achieved "whereby no additional data could be found where the researcher could develop more properties" (Glaser and Strauss 1967, p61).

Still, it was an organizational narrative (Pentland 1999) and if a gap was felt and the developers seemed to discuss something that had not been followed by the researcher, a number of steps were carried out to retrace messages to ensure that valuable data had not been missed. These steps included ensuring that all the relevant threads and messages were collected including simple repeated searches with keywords, the use of derivative keywords, cross-searching across other LKML archive sites because their search methods offered specific facilities, and following up a number of emails from the key protagonists even with seemingly unrelated thread titles just to confirm that they did not refer back to other themes of discussion.

Data Analysis

Table 3 depicts the process by which the data was analyzed. It broadly followed four steps. The sheer size of the data dictated that we conducted a smaller focused data analysis before heading into the entire text. In order to make this scaled-down data analysis less biased in terms of the sort of open codes that might emerge, the first step included a selection of 100 sample email messages (they were in sequence, but were not the earliest emails chronologically). The first author then developed descriptive concepts through open coding of these chosen 100 emails. The initial coding was done in line with the basic rules of Glaser and Strauss (1967). This was useful as far as generating ideas was concerned but the coder found that at the end of this one hundred emails open coding there were multiple and overlapping codes. These were then evaluated by both authors and crosschecked against the entire code book bringing the tally to about 100 mutually exclusive open codes that we could all agree upon. Having built the codebook

the first author then applied it to code the rest of the data. This stage took a fairly lengthy time but at its end there were over 150 mutually exclusive descriptive concepts.

In Step 2, both authors created a timeline together to break down coordination into events as we saw it shift and change at different moments of time (Strauss and Corbin 1998). In all cases in our story we found such events to be centered on controversies of *which* version control software to adopt, and *why* (Strauss 1987; Strauss and Corbin 1998). Each event was either something that transpired (occurrence) or an event that was supposed to happen and was expected by the developers, but for some reason did not occur. In both cases it led to debate and discussion where a serious flurry of emails moving back and forth could be followed. The timeline was constructed and then crosschecked against the mailing list repository for accuracy and discrepancies. The first author was more familiar with the data but the second author forced the first one to remain objective by constant questioning of each event, and if it did indeed qualify to be included. Twelve main events were found but the first and the last did not technically fall within the period of our study so we focused on the ten in the center to demarcate the main coordination events.

To qualify as an event, the following conditions had to apply:

a) An occurrence/non-occurrence that is considered controversial by the developers in relation to co-ordination,
b) Creates a sudden and increased flurry in email responses and discussion, and,
c) Something that has causal implications for the next event – i.e. is seen to be an input or catalyst for the next event.

Step 3 involved the identification of coordination processes. This entailed a process of clustering and identifying different types of relationships between the open codes – creating axial codes (Glaser 1992; Strauss and Corbin 1998). The first author made use of the memo facility in Atlas.ti to capture conceptual notes throughout this process of axial coding. Conceptual notes that were created built on current literature in coordination and drew upon dynamic coordination ideas (Harrison and Rouse 2014; Jarzabkowski et al. 2012; Kotlarsky et al. 2014; Langley et al. 2013). The data analysis gave rise to close to 30 theoretical memos on the issue of coordination processes and dynamics. It was through periods of

intense discussion between the authors that these memos were refined. They were not reduced by number but each memo took on fuller shape with the second author guiding the process of looking for links to literature in the same area. This led us to the idea of dynamics of coordination because our data nudged us sharply to notice how fluidly coordination moved at certain points of the Linux Kernel development. However, this was partly made possible by the ability to discern more than one coordination process. The process of coordination was fluid and dynamic both in its movement but also in its shape and essence. Returning to the literature on coordination and authority structures both authors looked for particular characteristics that set one type of coordination process apart from others. Open source literature emphasizes the existence of the meritocratic coordination process (Capra et al. 2011; Demil and Lecocq 2006; O'Mahony and Ferraro 2007; von Krogh and von Hippel 2006). Other forms of coordination processes such as autocratic, which are found in more traditional organizational settings (Eisenhardt 1985; Ouchi 1980), are not often cited in open source literature. Our work combines both to provide a more complete understanding of different coordination processes made possible in digital settings that abound today. Another noticeable difference in coordination processes was our discovery of a more nuanced breakdown in coordination types than is evident in literature. Our memos and axial codes coalesced around four coordination processes; autocratic clearing house, oligarchic recursion, federated self-governance and meritocratic idea testing.

Our final step, Step 4, urged both authors to trace the journey of each coordination process over the eight year period of our data. The process of disengaging one coordination process from another was not straightforward. It led to a discussion between the authors and a change in data analysis strategy. Both authors decided to work side by side but with their own computers. Focusing on the same data section we began to note the point at which any one singular coordination process was obvious. We approached the data through analytical time-slices to narrow down on the dynamics of coordination. It was during this stage of data analysis that we noted the existence of *multiple coordination process* swapping in and out of

focus at the same time. Coordination behaved very much like a fluid where intertwined coordination processes moved, changed, were fleetingly visible and sometimes disappeared completely. We mapped these movements across different event periods of our data to make sense of digital coordination.

Table 3. Data analysis

Steps	Tasks	Outputs
1: Open coding	a. Develop early descriptive concepts by coding 100 messages from data b. Re-evaluate and reduce the number of descriptive concepts by eliminating duplicates and merging closely related ones. c. Use this code book to analyze the entire data material	• There were close to 100 mutually exclusive descriptive concepts created from the pilot • Ended with over 150 open codes when open coding of the entire data material was complete
2: Timeline of key events	a. Establish a timeline of coordination events, version control changes, and points of contention raised about each VCS b. Cross-check timeline with mailing repository, and each link	• A chronology of key events (see Table 4) • Process of coordination analytically decomposed into different processes evident in the data
3: Identify coordination processes (CP)	a. Clustered open codes and laid bare relationships between them to build axial codes. This involved examining, categorizing, and clustering concepts. b. Memos emerged naturally during the course of coding. Conceptual ideas related to the data (and linked directly/indirectly with literature) were noted in some detail c. Elements of overlap and difference with extant literature on coordination processes gave rise to recognition of already known coordination processes but also the ability to discriminate new ones d. Our four coordination processes were categorized and named	• Built close to 30 theoretical memos[1] related to coordination processes • New coordination processes such as autocratic clearing, oligarchic recursion, federated self-governance and meritocratic idea testing. Codes and memos coalesced around each CP indicating the defining characteristics of each.
4: Distinguish concurrent coordination processes	a. When establishing the time-specific appearance of coordination processes proved frustrating due to a 'moving target' phenomenon the analysis was reversed to seek liminal coordination processes b. We used the notion of analytically slicing the data into time-frames to slow down the process of coordination movements c. This lead to the identification of a new dynamic of coordination where multiplicity of coordination was evident – strands of different coordination processes were followed in the emails to re-experience their separate moments, their intertwining, moments of liminality to disappearance even	• Identification of moments where the existence of multiple coordination processes were evident • The liminal nature of coordination where different coordination processes were made visible

[1] The theoretical memos that arose from the data analysis covered the main themes of the data, including that of coordination processes. They were deep and yet broad enough to span the entire theoretical apparatus we found in our rich data.

FINDINGS: THE LINUX KERNEL DEVELOPMENT

In 1991, Linus Torvalds started what is now typically seen as the most significant open source development project: the Linux Kernel[2]. In the first couple of years, the project was small and essentially managed by Torvalds, using simple tools such as his university email account. However, the versions kept changing and the number of developers and contributions grew. For instance, between December 1991 (version 0.11) and July 1995 (version 1.2), the number of contributors grew from three to more than 15,000 (Moon and Sproull 2000), making the open source project, and its outputs in the form of software code, increasingly complex to coordinate.

From the outset, Torvalds managed to create and sustain a deep respect within the community of developers, in which he had been described as the "benevolent dictator the whole community trusts" (Larry Augustin, 2001). Yet, over time, the sheer complexity of managing the rapidly scaling software offered many occasions where this ownership was contested, and showcased a range of coordination options that were explored with mixed success. In what follows, we narrate the eight-year evolution in coordination dynamics in the development of Linux Kernel software by zooming in on the main version control events between 1995 and early 2004 (see Table 4). There are ten main events that we focus on and they are clearly laid out in Table 4. After a brief historical narrative that provides background to our study we then develop the coordination processes at play within the Linux Kernel project.

The Early Years (1991-1998): Early on, coordinating Linux was seemingly simple. Although a software tool for version control called Concurrent Versions Systems (CVS) was in use by UNIX developers, the Linux project did not really merit such a tool, since it still lacked the critical mass of developers. In fact, Torvalds was on top of much of the coordination, including making pre-patches and/or releases. Increasingly, however, trusted individuals, referred to as maintainers in the community were appointed to

[2] The kernel is a computer program that is the core of a computer's operating system, with complete control over everything in the system (Wikipedia, 2017).

serve as gatekeepers. This helped to protect Linux code and maintain "his tree"[3] in the manner that was closer to the liking of Torvalds.

Even though the group of maintainers had grown to at least a few hundred[4], in 1996, the sheer volume of patches made coordination problematic. A number of senior maintainers therefore voiced the need of a version control tool. One of the most trusted maintainers, David Miller, even started to maintain a version of kernel code, which he labelled VGER (pronounced "Voyager"), to support version control. Apparently, this was significantly needed, since many developers started to use it as a point of reference in their software development work.

However, Torvalds' influence served as counter-force as he refused to synchronize Linux code through VGER (Torvalds, 28th Sept 1998). His refusal was met with resistance from a section of the community that was led by Miller, who in turn stated clearly that he was not going to disable VGER or stop using it for Linux development (Miller, 29th Sept 1998). This was a striking example of how maintainers, using version control software, began to take on more responsibility than originally intended. Also, it exemplified how new and more fine-tuned forms of coordination began to emerge. This was necessary because both software patches and the developer community were increasing in size. However, so was the complexity of the Linux Kernel code, making new forms of coordination yet more critical that would help to cope with the growing complexity.

While showing interest in the idea of version control software in general, Torvalds was decidedly against the use of CVS in particular. CVS was used by a subset of Linux developers but Torvalds rejected its use for his own work on the kernel. This resistance provoked a small but persistent outcry in the community: The developers wanted Torvalds to pay attention to their patches and to provide them with one definitive tree of source code to pull from. It was a simple coordination issue, they argued, which any

[3] In open source development only the patches you add are owned by you (under the GPL and other OSI licenses) but Torvalds was known to call Linux his project. His 'tree' refers more specifically to the actual branch of Linux that he maintained and controlled.

version control tool could ameliorate. Coordination tools such as version control software could afford the entire community more transparency and awareness of progress on any version and releases.

Roughly at this time, in late 1998, a number of other version control solutions were proposed. Each of the tools promoted by one or many developers were discussed with regard to the functionality offered and the needs of Linux development. Still, it was a different version control software, BitKeeper (BK) that would soon become a point of both controversy and interest. Its creator, Larry McVoy had often thrown in some minor comments seeking feedback from Linux developers as to what would be acceptable to them in version control software. The developers were generally aware about his closed, and proprietary tool under development, and showed their hesitancy about such license mixing. McVoy was still eager to please the developers because he needed them to test the software, and even offered to bring BK under an open source license if the parent company got bought.

Table 4: Summary of Events

Events	Event Description
1: June 1995 **Email coordination of community work**	• Single leader/coordinator • Growing community of developers • Use of simple, but multiple tools for coordination • Torvalds unable to cope with all the submissions • More newbies than experienced developers joining so a greater need for guidance • Growing need for more sophisticated control and coordination
2: September 1997 **Introduction of VGER use**	• Team of trusted lieutenants • Some use of a branch of CVS, renamed VGER, by a sub-section of the community • Bug fixes rising in number and complexity • Conflict and overwrite caused the use of disparate tools to update the Kernel
3: July 1998 **CVS coordination refused**	• Torvalds begins search for comprehensive tool • Strong body of developers emerging as a community • Yet patches being ignored • CVS use dismissed by Torvalds • Deepening impatience of the community with Torvalds
4: February 1999 **BitKeeper introduced**	• Closed source version control software introduced to the Linux Kernel community • Passionate resistance evident at the thought of a closed source tool adopted for coordination • The alternative of NO version control was just as untenable so other options explored
5: September 2000 **Linux Kernel Patch Penguin**	• Linux Kernel Patch Penguin considered • Threats of forking • Linus not scaling to corral agreement on a singular solution • Torvalds dropping patches and enraging the community • Some senior developers rallying to find a singular, efficient solution proving difficult due to disagreements
6: February 2002 **BitKeeper adopted**	• Torvalds makes the bold decision to adopt BitKeeper • Unease as the developers come to terms with decision to adopt a closed source tool

	• BitKeeper's introduction creates almost two clear factions in the community – so though BitKeeper makes coordination more efficient, it at the same time slows momentum through generating discord
7: April 2002 **Change in BitKeeper license**	• Linux Kernel community dismayed to see a regressive change in BitKeeper's license • Change in BitKeeper's license brought the realization to even the more sanguine developers that circumstances may well deteriorate • A change in BitKeeper's license rouses the community. Their mutual, and vociferous indignation begins to unnerve even McVoy • Many voices suggesting that the time for a replacement for BitKeeper has come and needs to be acted upon
8: February 2003 **BitBucket announced**	• An open source copy of BitKeeper called BitBucket announced • Questions of copyright and ethicality of a BitKeeper clone debated to make sense of the tool needs for the Linux community • The Linux Kernel community is a good testing ground for BK so McVoy is loath to make a drastic move BitBucket forces him to make threats. • McVoy determined to keep the community without changing the license conditions of BK looks for another solution.
9: March 2003 **BitKeeper to CVS gateway announced**	• Anxious over possibility of BitKeeper replacement McVoy announces a technical gateway between CVS and his own tool BK • Generally the community acknowledges that a tool is VERY necessary for coordination and control of the Kernel's work • Creation of the gateway eyed very suspiciously by the community • Community is not convinced that it can access 100% of the Linux Kernel material through the gateway thus bringing its integrity, existence and motivation into question • When discussion does not prove to change McVoy's ability to provide full access to the Linux code then the community turn to resistance.
10: July 2004 **BK→CVS gateway breakdown**	• Mistrust of McVoy and his company spreads to the gateway he has created for CVS users to access updates on Kernel development • Few, if any developers use the gateway after harsh criticism of its integrity and providing full access and control to all the community • The community not only resists BK control through a lack of use, it also begins multiple proactive changes (which are only known to a few at first) • BK is reverse engineered by some members of the Linux Kernel community compelling McVoy to remove BK from community use. This breaks the relationship between BK and Linux. • Torvalds recognizes that a good tool is even more pertinent now decides to abide by the open source way – and scratches 'his own itch' by initiating GIT.

BitKeeper (1999-2002): In early 1999, the need of version control software was widely recognized, but the more crucial discussions flowed from the controversies around which tool to adopt. There were a number of unofficial trees growing, which added to the coordination problems already faced by the developers. Developers were no longer sure as to which version control tool had the most up-to-date version of the Linux Kernel, and which branches had been changed and where. Developers saw a duplication of contribution as a real waste and there was growing frustration among the community with Torvalds. Indeed, there were clear signs of Torvalds' incapacity to manage version control. First, there were fewer new patches being submitted for peer review. In addition, there were many more resends. Developers were

forced to send the same patch for peer review numerous times before it was acknowledged, let alone reviewed. As a result, the community of developers began to fracture with a growing number of dissenting voices. Linux Kernel development was at a critical juncture, and in great need of a good version control tool. McVoy sensing discord and opportunity brought a new version control tool, BitKeeper, to the community's attention.

Although Torvalds was aware of the risk that his developer community would take their code and talent to another project, he was still not keen to accept BitKeeper without stronger persuasion. Indeed, while considering BitKeeper use, Torvalds was also busy working on his own solution called the Linux Kernel Patch Management System, launched as a project proposal on September 13, 2000. Despite much initial interest, however, discussion amongst the developers steered off into a comparison of CVS and BitKeeper rather than showing any genuine interest in the Linux Kernel Patch Management System. It was now obvious that if the Linux Kernel project was to remain sustainable, Torvalds needed to appreciate developer discontent and go with another solution. Accordingly, in February 2002, Torvalds declared BitKeeper the official version tool for the kernel.

However, many in the community did not appreciate BitKeeper adoption due to its non-GPL license. In early October 2002, the situation grew worse as some developers noticed that the BitKeeper license had been changed to include a new clause, '...this License is not available to You if You…develop, produce, sell, and/or resell a product which contains substantially similar capabilities of the BitKeeper Software'. The substance of concern about BitKeeper not being GPL was expressed by another developer, Molnar who clarified how the Linux Kernel code was both the source code and the developer comments (metadata) and though the source code was under an open source compliant license (GPL), the metadata was not (Molnar 2002 - Sun 6th Oct). The use of more than one version control tool at the same time by the Linux community offered the ability to hide, hold, withhold, and manage both the developers and their code – in essence, it was increasingly difficult to distinguish a clear locus of coordination in the Linux Kernel project. The multiple version control tools in use made it difficult to distinguish where coordination activities originated.

BitBucket (2003-2005): While BitKeeper slowly began to gain ground, this momentum was partly broken when Pavel Machek (another Linux developer), in February 2003, started a project called BitBucket. The project was framed as '(a) *bitkeeper clone*' (Machek 2003 - Wed 26th Feb). Not surprisingly Larry McVoy retaliated by telling Machek and the Linux community more generally that BitKeeper was a trademark (McVoy 2003 - Sat 1st March). The announcement of BitBucket generated opposing views. On the one hand, BitBucket was seen as a source of fragmentation in coordination. On the other, certain developers argued that competition was necessary and that such competing coordination tools should emulate the current tool being used in order to ensure a smooth changeover.

Torvalds believed that the key feature of BitKeeper was distribution rather than the repository format. He explained that if two people, or more, would give the same name to their similar or different files, then merging becomes very tricky. Distributed repositories imply that any developer that pulls code from the tree in effect creates his own version, which can be renamed, and changed in other ways. Pushing the changed version back into the tree can create serious complications because there will now be multiple versions and no way to decide which one is better or more useful. With a version control tool which has a central repository, like CVS, this is not a problem because conflicts in file naming and multiple commits needs to be resolved before the final commit is made. The same was not true for BitKeeper, a distributed version control software.

However, perhaps McVoy felt that he had made little headway with the community at large. Less than a month later, he surprised developers by creating an interfacing gateway between CVS and BitKeeper (McVoy, 2003 – Tue 11th March). While gateways in software are seen as very useful ways to access external code, at the same time they provide little incentive to any developer to create a competing product. The BK to CVS gateway provided indirect access to BitKeeper to all the community, making working on a competing tool unnecessary. However it did allow CVS users more access into what was happening in Linux development.

The slender trust between the open source developers and the proprietary BitKeeper, in spite of McVoy's move to create the gateway, was challenged yet again. Ben Collins expressed the main concern

in terms of not gaining access to all Linux data but only about 90% of it, a fact that made the community very uneasy (Collins 2003 - Tue 11th March). Larry McVoy insisted that nothing was missing from the data and that some of the developers were just being paranoid and mistrustful.

In March 2003, Larry announced that the BK to CVS gateway had gone live and could be accessed and used but the gateway was not maintained for long due to '*disuse and security problems*' (Anvin 2004 - Thu 22nd July). A certain momentum had been building slowly but compellingly that indicated multiple concerns with BK functions, form and use. The Linux Kernel developers were a strong community of creative and skilled programmers who believed that the open source way was to always try and find a solution to their own problem. They did not appreciate BK adoption despite its strength as a version control software. In April of 2005 McVoy accused some members of the Linux community of attempting to reverse engineer BK. He pulled BK use from the Linux Kernel collective. In response, Torvalds decided to create his own version control software, which he named GIT.

ANALYSIS: COORDINATION PROCESSES IN THE LINUX KERNEL

The story of Linux Kernel development shows a dramatic and sustained growth in developer numbers and their contributions. Growth of such proportion necessitated version control that both preserved the Linux Kernel's installed base from being fragmented and effectuating new contributions by developers. The struggle over eight years shows a journey of trial and error with different tools for ensuring that patches were managed, problems were flagged, multiple versions were possible, and large amounts of time-stamped data were held and archived. What is evident from our story is that each tool was actually sampled by the community for accomplishing specific forms of coordination. Such coordination forms were then amended and tweaked to become a better fit for the ambitions of this growing open source project. In most cases, the tweaks proved insufficient and the tool was eventually dropped. The adoption, tweaks and dismissal of tools were revealing of the changing and dynamic nature of coordination needs of such a large and eclectic

community. Our results indicate four unique coordination processes prominent in the Linux Kernel development project. In what follows, we discuss autocratic clearing; oligarchic recursion; federated self-governance; and meritocratic idea-testing.

Autocratic Clearing

Our data analysis (see Figure 1) showed that a powerful basis for coordinating the development of the Linux Kernel involved what we refer to as *autocratic clearing.* Autocratic clearing is a system of management with singular coordinating points that obliges other actors to channel all work and decisions through a central 'clearing house' before accomplishment. First, this coordination process involves a *singular point of entry and exit* for access to significant resources for coordination. This means that the actor who controls this point of entry will gain significant influence over the development work. In our case, it was clear that what attracted Torvalds to any version control software, be it simple use of e-mail or more sophisticated tools such as BitKeeper, was the provision of such a singular point of entry. For instance, e-mail was eminently controllable by Torvalds, as he was the only one privy to his account. Second, it also involved *centralized decision-making* where significant decisions for the future of the software were concentrated to a single actor. In the Linux Kernel case, Torvalds made many of the important decisions about patch acceptance and version releases. Finally, such decision-making was supported through a technical infrastructure that served as intermediary between proposed contributions and new releases of the software. Such *clearing* exhibits market-like qualities where patches are exchanged and valued (and evaluated) by developers. In our case, Torvalds was the regulator of this clearing-house, and could step in to effect a decision, thus governing this marketplace of ideas.

<< Insert Figure 1 about here >>

Autocratic clearing appeared early in Linux history but its effect was felt throughout the study period, and beyond. Centralized decision-making remained a strong idea in Linux development, and the choice of version control coordination tool largely depended on the level of centrality and ownership offered to the leader. Consider how the decision to either adopt CVS or BitKeeper essentially was one of sustaining a single entry and exit point for all changes to Linux code. While CVS had created multiple

versions that were worked on in parallel, BitKeeper represented a way to accomplish a singular point of entry and exist in the context of a much larger community of developers. Viewed from an autocratic clearing perspective, parallel versions are essentially a sign of disorganization. Distributed version control requires that all the developers that have write access should possess a degree of expertise but this is not easy to engineer.

Whereas single point of entry and exit was a quality that could be designed into a digital tool, keeping all decisions to be made centrally and as much as possible by Torvalds was increasingly difficult as the open source project grew. Clearly, more effective and new coordination techniques emerged in view of the need of a resilient project.

The use of BitKeeper restored autocratic clearing especially compared to the e-mail system used as the early version control of Linux Kernel. While BitKeeper seemingly gave multiple entry and exit points, in practice however, multiple access and central decision-making rights were not given to any one developer at the same time. If a developer was able to pull content and create a personal clone of Linux, and make changes to his cloned version, this new and changed cloned version could not automatically be pushed back into the main build of Linux. This is where BitKeeper supported centralized decision-making and served as a technical clearing-house in coordinating open source development contributions.

Oligarchic Recursion

The second coordination process generated through our data analysis was that of oligarchic recursion (see Figure 2). We define oligarchic recursion as a system of management where decision-making is stratified and the aim is to strengthen the current controlling structures through recursive handling of code peer evaluation. For a long time effective coordination meant reliance on a group of tools and actors working together cooperatively to manage Linux development effectively. This involved *stratified decision-making* enabled by extra layers of design embedded into the digital tools to make coordination fractionally more diffused yet still cordoned off into the control of a few actors. These actors were often close allies and shared the same mental models of coordination or design specification. For instance, Patch Penguin, considered as version control software in 2000, was supposed to embody a two-tiered development

style provoking parallel versions, where different maintainers could oversee each one. This stratified development, the key decisions-makers, and how the stratified versions would be brought together. The two branches of similar code were worked on simultaneously but handled by different coordinating maintainers. Yet, at some point, there was convergence of one branch towards another where both versions were either (a) merged or (b) assessed for different use value and broken off to fulfill completely different functions. Both the convergence and ability to work on multiple branches concurrently was made possible by digital coordination tools like version control software.

<< Insert Figure 2 about here >>

Second, patches submitted by open source project members went through a *formalized form of evaluation* before acceptance/rejection. Since the Linux project grew larger in code base size, developer numbers, and complexity, evaluation of patches required different forms of scrutiny to avoid the danger of having software updates overwritten or ignored. Various extra layers of management were therefore introduced into Linux evaluation to ensure the safekeeping of code updates. The stratified nature of decision-making embedded into technology offered a comprehensive and rule-based process of evaluation. With a set of routines and rules in place, the maintainers could help make decision-making more efficient. Each maintainer, through privileged access to branches in the version control software, evaluated and decided on different segments of code. The process of evaluating software, though always a key part of open source development, had now been made far more formalized with distinct stages and routines to be followed by both the maintainers and the developer community.

And finally, there were mutually cooperative structures, practices, rules, and tools purposefully built to *reinforce authority recursively*. Designing coordinating power into technology was not as complicated as keeping it from obvious view of the community. Technology is often embedded with rules and structures that are seen as part of good design and requirements, yet in practice users are less able to notice such rules unless they face them as a possible restriction in use. Oligarchic recursion established the

powerful actors, and further entrenched them into the Linux project through instituting nested and recursive practices for code approval, shortlisting of patches or developers for different roles, or the choice of new maintainers. The fact that BitKeeper only allowed privileged actors to "cherry pick" patches from the main tree to accept/reject was not a facility offered to the rest of the community of contributors, and furthered the needs of the already established elite.

Federated Self-Governance

We refer to the third coordination process in practice within the Linux project as *federated self-governance* (see Figure 3). We define federated self-governance as a management structure that establishes semi-autonomous governing bodies that work together, making sovereign decisions until a re-merge is made necessary. In the Linux Kernel case, federated self-governance emerged in response to patches not being updated and accepted fast enough. One manner of achieving this was to break future code into different branches to create *semi-autonomous governing bodies*. We saw a benign and internal forking of Linux into three different branches, all done to cater to the diverse needs of companies and developers. A stable, beta and alpha branch of Linux emerged and a federated form of coordination to match. Each branch was developed at a different pace and held slightly different code. Maintainers, with a group of dedicated developers, governed each branch of the code. Decision-making for each branch was semi-autonomous because each branch would be left alone to grow and change according to the needs of the developers working on it, and the maintainer who was in charge with little or no interference from Torvalds and other maintainers.

<< Insert Figure 3 about here >>

The creation of self-governing structures was imperative to maintain growth in Linux. However, to periodically draw the branches together, to *remerge*, was equally important. The version control software was used to update the stable branch, differentiate and unpack the needs for the beta one, and reestablish the tested (alpha) into the stable or beta branch as necessary. This was done to ensure compatibility and sometimes to introduce elements into the stable branch from the beta or alpha that no longer required extensive testing (and thus fell more naturally into the stable category). Seen as an essential moment in the

various cycles of Linux such coordination often brought about a new release and version of the Linux/GNU operating system. This is an example of a software driven need to reassess the underlying design norms and code. The branches are usually different but not completely unique, and thus can be merged to make certain functionality more effective.

Meritocratic Idea-Testing

The final coordination process evident in our data analysis was that of *meritocratic idea-testing* (see Figure 4). Meritocratic idea-testing embodies a style of management where mutually agreed decisions able to accommodate multiple divergent views are negotiated in a transparent manner so as to enable open questioning and testing of ideas in order to make them strong. The salient categories that arose pointed to firstly a more *communitarian decision making* amongst equal actors that technology facilitated and made possible. Secondly, the aim of this diverse way of making decision was to *achieve an accommodation of multiple views* between different developers. Thirdly, meritocratic coordination entails the need for technology to reinforce *transparency* into coordinating work so that all the developers can see the most current changes, the process of work, and older versions to check for potential problems. And finally, *stress-testing of ideas* is a crucial element of meritocratic coordination where multiple "eyeballs" (developers) can be put to any patch and problem to recognize the problem quickly. This allows good code to emerge efficiently as finding a solution to a problem becomes that much easier when the problem is clear and known.

<< Insert Figure 4 about here >>

Communitarian decision-making was clearly manifested in the use and adoption of CVS by the Linux community. CVS gave cloning access to all, and at the same time it offered all developers an equal chance to submit their changed code back into Linux. Decisions at the point of conflict between two or more patches sent to resolve the same issue would be made by the last developer to upload his/her change. Version control software such as CVS ensured that all developers were equal in its eyes – meritocratic coordination structures were embedded into its very design making it possible for all developers to feel included and their code valued.

An accommodation was necessary in meritocratic structures because all the voices had to be heard but of course could not be acted upon at the same time. The Linux story is littered with examples of accommodations found between different viewpoints; indeed each requirement in an open source project is a consequence of communitarian deliberation and accommodation. The development of the BitKeeper to CVS gateway is an example of how certain actors attempted to find an accommodation between the BitKeeper and CVS users. If a developer was not allowed to use BitKeeper or did not want to then he could still work on Linux development so long as he had access to the code. The most updated versions of Linux were held and controlled by BitKeeper so it was important to access this software. The gateway was created after long negotiations amongst the community and made operable as an accommodation to hold the community together over Linux code.

The creation of Linux Kernel Patch Penguin represented an attempt to establish meritocratic idea-testing. The aim was to rebuild transparency into an open source project that had been dissipated through a use of non-open source software for coordination. The very design of this version control tool was being planned to reinforce transparency and to make communitarian decision making possible. This tool may never have actually materialized yet a discussion of its possibility was central to the debate of what sort of version control software the Linux community preferred. It was indicative of their needs but also their preferences. And though it eventually turned into a discussion and comparison of CVS and BK, it did create a much needed awareness of the strengths and weaknesses of both these tools. The debate was part of the process of true meritocratic coordination where stress testing of ideas is carried out through debate in a transparent and inclusive manner.

Stress testing of ideas is a basic premise of open source development. Linux development was no exception, and it is evident in numerous examples where individual patches were scrutinized to evaluate strength, functionality, and elegance. But it was not only the code that was stress-tested in Linux. The very process of software development as routinized within the development practices of version control

software are also tested for their ability to cope with different issues such as the growing number of software commits being made at one point of time, or who is designated as the final stress tester of ideas, and has the power to make a decision between two or more patches submitted to solve the same problem.

Coordination Processes Coming Together

We distinguished four coordination processes (autocratic clearing, oligarchic recursion, federated self-governance, and meritocratic idea-testing), each displaying common sets of rules, guidelines, and activities that operationalize a specific authoritative structure. However, as our case story unfolds, it is clear that the coordination processes were not standalone, plotted in a sequence replacing each other. Instead, they co-existed at certain moments in the evolution of the Linux kernel. Two or more coordination processes were typically active at any moment, sometimes offering complementary support by addressing different coordination needs in the development of the software. Other times the co-existence spurred tension as the coordination processes collided. In other words, in addition to the direction unfolding within each of the coordination processes over time, there also existed a dynamics as coordination processes were coming together in the governance of the Linux Kernel. Table 5 summarizes the four configurations that we highlight here as representative 'moments', and each one is explained in the following section.

Table 5: The Configuring of Governance

Configuration 'Moment'	Coordination Processes	Complementary Support	Opposing Tensions
A-M-F (VGER Shutdown)	• Autocratic clearing (A) • Federated self-governance (F) • Meritocratic idea-testing (M)	Autocratic clearing (A) and meritocratic idea-testing (M) gave complementary support to each other to help create transparency – much needed in open source development processes.	At the same time, federated self-governance (F) emerged through developers enacting their persistent use of VGER as their choice of version control software. This reduced transparency overall but at the same time allowed voicing of self-expression and actual progress in development work.
A-O-M (Introduction of BitKeeper)	• Autocratic clearing (A) • Oligarchic recursion (O) • Meritocratic idea-testing (M)	BitKeeper succeeded in instituting subtle emergence of oligarchic recursion (O) alongside autocratic clearing (A) and meritocratic idea-testing (M) where all these forms of processes eventually *worked in tandem* to manage synchronized work.	
A-O-M (BitKeeper License Change)	• Autocratic clearing (A) • Oligarchic recursion (O) • Meritocratic idea-testing (M)	The complementary support of autocratic clearing (A) and oligarchic recursion (O) was an effective way to manage different	The license change for BitKeeper created barriers for certain Linux developers because they were no longer allowed to use tool. The relative harmony

		factions of the community, some of which were not happy with the use of BitKeeper. However, a change in BitKeeper's license disrupted smooth coordination.	of autocratic clearing and oligarchic recursion at work through BitKeeper was then disrupted by a substantial number of developers who decided through meritocratic idea-testing (M) that a new tool was needed.
F-M (The Introduction of BitBucket)	• Federated self-governance (F) • Meritocratic idea-testing (M)	BitBucket managed to establish complementary support between federated self-governance (F) and meritocratic idea-testing (M) because it could encase and capacitate multiple forms of synchronization within and across coordination processes.	

Configuration 1 (VGER Shutdown): In 1995, autocratic clearing, federated self-governance, and meritocratic idea-testing came together in the governance of the Linux Kernel. This was the time when Torvalds demanded that VGER be shutdown. In acting as semi-autonomous governing bodies (federated self-governance), groups of developers chose to use a variety of version control tools, among which VGER was the dominant choice. This enactment of federated self-governance caused friction, since Torvalds was still keen to exercise his moral ownership over the Linux Kernel project. VGER was a threat to the sets of rules, guidelines, and activities that operationalized autocratic clearing, especially the aspect of singular point of entry and exit, which was essential for putting clearing in the hands of Torvalds. Yet, Torvalds' demand was vested in a mix of autocratic and meritocratic structures. Albeit adjudicated by an autocrat, meritocracy offered complementary support as transparency was concurrently relevant, and Torvalds felt that VGER, and the developers' enactment of semi-autonomous governance hindered such transparency. This is evident from the exasperation expressed by Torvalds when he could no longer account for which version was the most updated in Linux as various branches had been given different updates so that the community was left with no coherent coordinating control. In the time that followed the interaction between the three coordination processes (autocratic clearing, meritocratic idea-testing, and federated self-governance), in which VGER's role in the Linux project was contested, we found that the complementary support of autocracy and meritocracy helped discontinue most of its use but this left a vacuum in efficient coordination. The community reverted to the use of multiple tools to manage work but this was not efficient, and led to more discussion about a possible custom built tool for the entire community. The three coordination

processes had operated as both complementary and as a source of tension as they were intertwined in the VGER shutdown. The vacuum was felt due to an inability to decide on which form of version control and coordination process would work for governing the entire community. This is not surprising considering the diverse actors involved in the project.

Configuration 2 (Introduction of BitKeeper): As BitKeeper was introduced in 1999, autocratic clearing, oligarchic recursion, and meritocratic idea-testing were active and intertwined in their tussle for supremacy. BitKeeper offered a singular point of entry (autocratic clearing), and Torvalds' eventual support of the version control software can be seen as evolution of autocratic clearing in the context of the increasing amounts of developers and updates to manage. While the community of developers both recognized Torvalds as a leader and had called for a proper version control tool for a long time, however, there was growing discontent with BitKeeper. This was not primarily or only related to the fact that BitKeeper coordinated the Linux community on Torvalds's behalf in its instantiation of rules and practices approved by Torvalds. Rooted in meritocratic ideals, developers rather reacted to how BitKeeper reinforced authority recursively. For instance, as BitKeeper supported oligarchic ideals by only allowing privileged actors to cherry-pick patches, and as it become increasingly clear that the meta-data generated was not owned by whoever created it, many developers rooted in meritocratic ideals did not accept the adoption of BitKeeper and turned to proxy tools to coordinate their work. Sometimes these tools were as simple as the use of email, while others looked to other version control software. However, when autocratic clearing and oligarchic recursion complemented each other and worked together then we notice in our narrative how on occasion this combination was able to subdue the emergence of meritocratic idea-testing. It can be surmised that as certain coordination processes worked in unison and compounded the effects of the other, they were better able to drown out opposing coordination processes that brought tension to the situation.

Configuration 3 (BitKeeper License Change): In the beginning of 2002, the new BitKeeper license accentuated the discontent with the coordination supported by BitKeeper, resulting in various types of coordination breakdowns such as more time spent arguing than sharing code, fewer decisions being made even in relation to the new license so that the progress of the Linux Kernel became hampered, and fewer

updates of Linux were released. Torvalds had maintained an autocratic clearing coordination process, which he had balanced with an oligarchic style of governance through the stratified decision-making of BitKeeper. However, this was severely challenged as the pronounced recursively structured authority (oligarchic recursion) of the version control software became clear through the change in BitKeeper's license. What was also becoming evident was how singular coordination processes were unable to cope with the growing complexity and size of the community and number of code patches being added and scrutinized.

The Linux project was a growing mix of different levels of expertise in developers, various branches, differences in code and coding practices and of course a diversity in ideology. Such growing complexity in online community work requires both a sophistication in tools and flexibility in authoritative management. What is noticeable in the Linux Kernel project is how decisions about the adoption of tools such as version control software embody sophistication and flexibility of holding multiple authoritative processes in balance. Some tools were sophisticated enough to embody coordination processes that at first glance could be seen as opposites or contradictory. BitKeeper is such an example as it supported multiple coordination processes. BitKeeper had to appeal in functionality and sophistication to actors aligned with coordination processes with different authoritative structures. It had a sophisticated push/pull method of making local copies of Linux, making changes and then attempting a push of the amendments back to the community. The method reflected its subtle capacity to juggle coordination structures simultaneously while ensuring the single point of entry and decision-making favored by Torvalds. BitKeeper embedded meritocratic values in that it made possible multiple copies of Linux to be pulled by all the community members. But privileged access and specified roles of developers determined the level of pushing code back into Linux. In the latter functionality, BitKeeper gave priority to autocratic and oligarchic coordination structures. However, BitKeeper's suitability was eventually questioned as the actors in the community began to unravel different coordination processes made available by it, and some developers of the open source project opted to find an alternative that was closer to meritocratic coordination ideals.

Configuration 4 (Introduction of BitBucket): Finally, in March 2003, the emergence of BitBucket involved an encounter between federated self-governance and meritocratic idea-testing. The moment, created in the wake of the never resolved divide related to the license changes in BitKeeper, was ripe for presenting BitBucket as a version control software based on federated self-governance ideals. BitBucket was supposed to offer the same coordination functionality as BitKeeper, but would come with the advantage of being an open source tool. In effect, this implied that semi-autonomous governing bodies (federated self-governance) could emerge where decisions would also become more self-governed, albeit later accommodated within the alliance. This was a call being met for more transparent development practices where the Linux community would return to open source roots of communitarian decision-making. Meritocratic idea-testing was concurrently present and actionable, for without community backing, Machek's BitBucket would not be sent any patches to coordinate and could then never become part of Linux coordination processes. Finding the delicate balance of transparency, solid functionality, and ability to cope with complexity was not straightforward. Machek was aware that building a rival to BitKeeper, a tool that had seen years of work and good testing ground with the Linux community would not be easy. He needed to play upon the ideology of meritocratic idea-testing to bring the community around to his new tool so that they would not only be willing to accept the possible breakdowns in it during its teething days but also help him to actually build it. What could be counted on in the case of BitBucket was that this tool, as it was being built from scratch would probably be designed to suit the coordination processes most appealing in effect and ideology for this community. Both federated self-governance and meritocratic coordination processes were evident because digital technology could encase and capacitate multiple forms of synchronization within and across coordination processes.

DISCUSSION

Our research set out to examine how coordination processes come together in governing open source software. Drawing on an in-depth, longitudinal study of the Linux kernel, we found autocratic

clearing, oligarchic recursion, federated self-governance, and meritocratic idea-testing as four distinct coordination processes. Each coordination process provided a common set of rules, instructions, and activities that operationalized a specific authoritative structure. We also found how these coordination processes co-existed in a way that provided complementary support in the governance of the self-selected contributors to the Linux project. In addition, they co-existed in a way that triggered tension and renewal. Authoritative structures reveal themselves in coordination processes when questions of authority over *access* are raised and decided (Boudreau 2010; Felin and Zenger 2014), when decisions are *interdependent* and cannot be resolved by the powers that be without affecting other aspects of open source development (Ben-Menahem et al. 2016; Crowston and Scozzi 2004), and when matters such as *task breakdown* (and re-mergence) are settled by the established order (Howison and Crowston 2014; Venters et al. 2014). Each coordination process is predicated on a specific authority structure that provides the foundational legitimacy for action. Such a connection is important to establish how coordination processes embody authoritarian structures, manifested in forms of code access, decision interdependence, and task breakdown.

Table 6: Implications for Open Source Governance

Open Source Governance Characteristic	Implications for Theory	Implications for Practice
Governance as a manifestation of multiple authoritative structures in open source projects	Literature on open source governance (de Laat 2007; De Noni et al. 2011; De Noni et al. 2013; Markus 2007; O'Mahony and Ferraro 2007; Tullio and Staples 2014) builds largely on a singular authority idea (Etzioni 1959). Our study contests this idea and substantiates our claim of multiplicity as one clear mechanism to make such diverse projects develop into successful communities over time.	Offers insight to companies that are engaged with open source communities to look to technology to embed governance rules into tools. Tools and software are better able to manage multiplicity of governance and coordination simultaneously, and can lead to more effective management of diverse motivations of open source developers overall.
Governance and coordination as a relationship of duality rather than opposite forces	Prior studies of open source governance (O'Mahony and Ferraro 2007; Tullio and Staples 2014) and coordination (Ben-Menahem et al. 2016; Crowston and Scozzi 2004; Howison and Crowston 2014; Koch and Schneider 2002; Lindberg et al. 2016) treated each phenomenon separately yet we challenge this separation and opposition of forces because our study shows how governance and coordination work together to reinforce each other. The horizontal workings of coordination constantly look to the more vertical governance for rules and norms that make the former executable.	Managers in companies that are tasked with open source community engagement often suffer when the community takes umbrage at being directed and orchestrated too much. Embedding both vertical governance processes as well as horizontal coordination rules into 'objective' technology can allow the company to better negotiate its relationship without suffering from ideological prejudice from the community.

We offer two main contributions to open source governance (see Table 6). First, we enhance literature on open source governance through the development of a novel theoretical perspective in which governance is seen as configurations of coordination processes, thus making governance multiple. Second, we conceptualize the relationship between governance and coordination as a duality. In what follows, we detail each contribution, and discuss its implications for our research.

Multiplicity of Governance in Open Source Projects

We extend open source governance literature (de Laat 2007; De Noni et al. 2011; De Noni et al. 2013; Markus 2007; O'Mahony and Ferraro 2007) with a novel perspective on governance. In particular, we show how coordination processes coexist, each grounded in a particular authoritative structure. While existing governance literature assumes governance rooted in a singular authority structure (e.g. Etzioni 1959) or views co-existing forms of governance as temporary (O'Mahony and Ferraro 2007), our perspective holds that multiplicity of authority structures is an important inherent feature of open source governance. Multiplicity is important because large-scale, distributed open source projects attract different sorts of developers that bring with them diverse motivations. These developers self-select to perform tasks (Hemetsberger and Reinhardt 2009; Lanzara and Morner 2003; Lee and Cole 2003) where it becomes necessary to have multiple ways to organize and govern them across distributed locations. This holds the open source community together by providing complementary support for different stakeholders (Aaltonen and Lanzara 2015; Eseryel and Eseryel 2013; Hemetsberger and Reinhardt 2009; Moon and Sproull 2000). At the same time, our findings also highlight that configuring coordination processes into a peaceful collective is not possible indefinitely and disruption is common. Ideological differences in the authoritative structures eventually surface, and serve as an important initiator of change in the governing of open source software. The success of distributed projects partly relies on balancing different, and sometimes conflicting authoritative structures simultaneously.

This contribution has significant implications. Open source literature to date has centered on singular authoritative structures regardless if the structure in question is centralized (Crowston and Howison 2005; Dahlander and O'Mahony 2011; Koch and Schneider 2002; Tullio and Staples 2014), libertarian (de

Laat 2007; De Noni et al. 2011; De Noni et al. 2013; Gallivan 2001; Howison and Crowston 2014; Raymond 1999), or collective (e.g. Hemetsberger and Reinhardt 2009; Markus 2007; Mockus et al. 2002; O'Mahony and Ferraro 2007; Shah 2006; Sharma et al. 2002). This body of work is largely in line with Etzioni's (1959) assertion that only one form of authority structure can exist in a given time. Even in cases where at least two authority structures have been observed (see e.g., O'Mahony and Ferraro 2007), this has been understood as a liminal event with an emphasis on how the authority structures soon converged into a singular, new form of governance. In our paper, we explain how different authority structures can co-exist as they serve complementary purposes across different coordination processes, and we emphasize that such co-existence is one of the crucial reasons for making complex coordination possible when the community sees different actors and technology as legitimate.

Second, we offer a perspective on how open source governance undergoes change. Our results suggest that understandably, complexity does indeed increase with growth in community size (Kuwabara 2000; Stanko 2016) yet where we differ from prior work (O'Mahony and Ferraro 2007; Tullio and Staples 2014) is in our finding that Linux revealed multiple coordination processes even when it was a fairly small, and simple project. We look to the crises in Linux development to explain the emergence of new orders of governance, coordination processes, and different configurations of governance. Some projects manage with a small, core team of developers (Krishnamurthy 2002) and never grow too big over time, yet they are vulnerable to various crisis points just as much as the large projects. And, while coordination processes, each with their own specific ways of channeling and constraining software development activity, can complement the other in a harmonized way, we also highlight moments when they disintegrate and reunite in new configurations that represent a different form of governance. Studies of open source governance (De Noni et al. 2013) have paid less attention to this aspect. We believe our perspective may serve as powerful basis for developing process models of open source governance as it offers a vocabulary that can be used in empirical studies designed to examine governance evolution.

And third, we show that digital technology (Lanzara and Morner 2005) plays an important role in allowing multiplicity in the authoritative basis of governance. Part of this is embedded in the design of the

technology and tools used by open source developers but we are also able to show through our Linux study just how such embedding occurs, with a justification of why, and to what effect. This is in contrast to a large body of the CSCW work where technology is seen as an unproblematic tool (Schmidt and Bannon 1992; Schmidt and Simonee 1996) rather than an artefact that is used politically by different actors to manoeuvre the community and the product being built. Our case illustrates how coordination processes were embedded in the software used for version control (cf. Cornford et al. 2010), and, intriguingly, that multiple coordination processes were simultaneously supported by the same software. We suggest the notion of multiple instantiation of separate coordination processes to capture this idea of digital artifacts embodying multiple, distinct sets of common rules and instructions with capacity to channel and constrain activity.

Our study also has implications for practice. Company engagement with open source projects over the last decade has grown substantially. However, such engagement has been fraught with discord and challenges on either side (Dahlander and Magnusson 2008; Dahlander and Magnusson 2005; Dahlander and Wallin 2006; Germonprez et al. forthcoming). Various managerial strategies have been employed by companies to build a healthier relationship with projects but it is less common to see them use technology to negotiate authority over the community-led project. This study offers valuable insight into how such technological tools can be designed and built to manage multiple ideological factions, and govern an open source project effectively.

Governance and Coordination as a Duality

Our second contribution offers a dualistic understanding (cf. Farjoun 2010) of governance and coordination in open source communities. Governance and coordination are separate yet at the same time they are deeply related processes that work together to produce change in open source communities. Prior literature has paid relatively little attention to the close relationship between the vertical dimension of governance (De Noni et al. 2011; Markus 2007; O'Mahony 2007) and the horizontal implementation of coordination (Crowston and Scozzi 2004; Howison and Crowston 2014; Koch and Schneider 2002; Lindberg et al. 2016). We can see this duality play out in every day practices and work in open source development. For

instance, vertical authority over access levels (Hemetsberger and Reinhardt 2009) indicates the relationship between what needs to be done and coordinated and how this is vertically delegated through access rights by the larger authority in a community. Authority over interdependence of decisions (Lindberg et al. 2016) reflects another aspect of where coordination meets governance through authority structures. There is complexity that emerges from decisions being interdependent on decisions made by others (and the status of authority that they carry). And lastly, authority over task breakdown (Dalle and David 2005) and possible reemergence of functions also necessitates a strong coordinating element of action and 'doing' but this action is always mitigated by the vertical authoritative capacity delegated by the more senior. Open source governance gives strong indication of vertical as well as horizontal movements[5]. These are not examples of processes working in opposition, but rather the coming together of two distinct mechanisms that together, effectively orchestrate an open source community.

Our study has implications for governance and coordination studies because we propose how these two constructs can effectively be used together to explain open source governance. The link between coordination and governance is more than hinted at in the management (Bruns 2013; Gulati and Singh 1998; Kellogg et al. 2006) and IS literatures (Markus and Bui 2012; Ribes et al. 2013; Scarbrough et al. 2014) but here we explore it in some depth and offer a detailed dualistic account of a longitudinal case where it is possible to reflect on the relationship between coordination and governance and their entangled relationship. While studies on open source governance focus on vertical authority (Felin and Zenger 2014), leadership (Conlon 2007; O'Mahony and Ferraro 2007; Oh et al. 2016) and other governance related ideas, the work on open source coordination negotiates more horizontal synchronization (Strode et al. 2012; Venters et al. 2014) and interdependence notions (Ben-Menahem et al. 2016; Howison and Crowston 2014; Lindberg et al. 2016) – but both literatures expand their contribution in siloed spaces. Our conceptualization of *coordination processes* personifies the deeply implicated relationship between 'who is able to tell others what to do' along with 'what needs to be done, and how'. In effect, we draw the

[5] We are grateful to a reviewer on this paper for drawing this to our attention.

two disparate open source literatures together to explain how these concepts work with each other in every day open source development.

The relationship of duality between governance and coordination also has implications for practice. Open source projects need to consider the tools that they appropriate more deeply (Shaikh and Vaast 2016) because the use of such technology is more than simply a question of either open or proprietary. The decision is far more complex because version control software can be used to build a community structure (or change it). Any and all tools appropriated to coordinate work have at the same time, serious implications for governance. The latter is not always understood or considered deeply and can have ramifications for the sustainability of an open source project.

Limitations

This study is not without limitations. First, we seek to offer an ideographic research explanation, viewing the findings as causal tendencies (Tsoukas 1989). This means that our research offers a perspective with which to examine the causal tendencies of the evolution and coming together of coordination processes in the governance of open source. However, this reliance on ideographic research explanation comes with the limitation that the study has a single setting and technology focus. Obviously, the study could benefit from comparative case analysis where different contexts of coordination processes are studied and compared to help theorization.

Second, in-depth interviews would have helped to build even richer accounts of the dynamics playing out in this governance setting. In particular, it would have been useful to collect interview data about how actors perceive forms of legitimacy manifested in the coordination processes.

Finally, our focus on a so-called "extreme case" (Gerring 2007) helps theorization to be "prototypical or paradigmatic of some phenomena of interest" (p. 101). However, this means that it may not be as representative of all open source projects at large. There are numerous open source projects that begin but fail within a few months. Coordination processes in such cases would no doubt take a different shape. Also, we cannot rule out that less comprehensive and public cases may be characterized by more stable, and perhaps singular, coordination processes.

CONCLUSION

Coordination processes and their multiplicity in an open source project are a significant and novel abstraction of how communities are governed. The idea that multiple coordination processes exist and work in tandem to effectuate a project over time goes beyond current literature to complement our understanding of coordination and open source community management. Drawing these processes together and orchestrating their multiplicity is the subtle link to governance. We put forward our thesis of the governance of open source software as a changing configuration of multiple and different coordination processes over time. Some processes fade away while others take hold more strongly. Digital technology used by open source communities, such as version control software, makes the multiplicity of coordination processes more possible. The technology that is able to effectively channel and effectuate multiple coordination processes becomes more acceptable and thus appropriated. We offer the research documented in this paper as a first step to further understand the evolution of governance in open source where multiple coordination processes are likely to co-exist.

REFERENCES

Aaltonen, A., and Lanzara, G. F. 2015. "Building Governance Capability in Online Social Production: Insights from Wikipedia," *Organization Studies* (36:12), pp. 1649-1673.

Adler, P. S., and Borys, B. 1996. "Two Types of Bureaucracy: Enabling and Coercive," *Administrative Science Quarterly* (41), pp. 61-89.

Alexy, O., George, G., and Salter, A. J. 2013. "Cui Bono? The Selective Revealing of Knowledge and Its Implications for Innovative Activity," *Academy of Management Review* (38:2), pp. 270-291.

Anvin, H. P. 2004 - Thu 22nd July. "Re: Linux-Kernel Cvs Gateway?", 2004, from http://www.uwsg.indiana.edu/hypermail/linux/kernel/0407.2/0490.html

Barrett, M., Heracleous, L., and Walsham, G. 2013. "A Rhetorical Approach to It Diffusion: Reconceptualizing the Ideology-Framing Relationship in Computerization Movements," *MIS Quarterly* (37:1), pp. 201-220.

Ben-Menahem, S. M., von Krogh, G., Erden, Z., and Schneider, A. 2016. "Coordinating Knowledge Creation in Multidisciplinary Teams: Evidence from Early-Stage Drug Discovery," *Academy of Management Journal* (59:4), pp. 1308-1338.

Bezroukov, N. 1999. "Open Source Software Development as a Special Type of Academic Research (Critique of Vulgar Raymondism)," *FirstMonday: Peer Reviewed Journal on the Internet* (4:10), pp. 1-23.

Boudreau, K. 2010. "Open Platform Strategies and Innovation: Granting Access Vs. Devolving Control," *Management Science* (56:10), pp. 1849-1872.

Bruns, H. C. 2013. "Working Alone Together: Coordination in Collaboration across Domains of Expertise," *Academy of Management Journal* (56:1), pp. 62-83.

Bryson, J. M., Crosby, B. C., and Bloomberg, L. 2014. "Public Value Governance: Moving Beyond Traditional Public Administration and the New Public Management," *Public Administration Review* (74:4), pp. 445-456.

Capra, E., Francalanci, C., Merlo, F., and Lamastra, C. R. 2011. "Firms' Involvement in Open Source Projects: A Trade-Off between Software Structural Quality and Popularity," *Journal of Systems and Software* (84), pp. 144-161.

Collins, B. 2003 - Tue 11th March. "Re: [Announce] Bk->Cvs (Real Time Mirror)." 2004, from http://www.ussg.iu.edu/hypermail/linux/kernel/0303.1/0894.html

Conlon, M. P. 2007. "An Examination of Initiation, Organization, Participation, Leadership, and Control of Successful Open Source Software Development Projects," *Information Systems Education Journal* (5:38), pp. 1-13.

Cornford, T., Shaikh, M., and Ciborra, C. 2010. "Hierarchy, Laboratory and Collective: Unveiling Linux as Innovation, Machination and Constitution," *Journal of the Association for Information Systems* (11:11).

Crowston, K., and Howison, J. 2005. "The Social Structure of Free and Open Source Software Development.," *First Monday*).

Crowston, K., Howison, J., Masango, C., and Eseryel, U. Y. 2007. "The Role of Face-to-Face Meetings in Technology-Supported Self-Organizing Distributed Teams," *Professional Communication, IEEE Transactions on* (50:3), pp. 185-203.

Crowston, K., and Scozzi, B. 2002. "Open Source Software Projects as Virtual Organizations: Competency Rallying for Software Development," *IEE Proceedings - Software* (149:1), pp. 3-17.

Crowston, K., and Scozzi, B. 2004. "Coordination Practices for Bug Fixing within Floss Development Teams," *First International Workshop on Computer Supported Activity Coordination (CSAC 2004)*.

Crozier, M. 1964. *The Bureaucratic Phenomenon*. Chicago University Press.

Dafermos, G. 2001. "Management and Virtual Decentralized Networks: The Linux Project," *First Monday* (11:6).

Dahlander, L., and Frederiksen, L. 2012. "The Core and Cosmopolitans: A Relational View of Innovation in User Communities," *Organization Science* (23:4), pp. 988-1007.

Dahlander, L., and Magnusson, M. 2008. "How Do Firms Make Use of Open Source Communities?," *Long Range Planning* (41:6), pp. 629-649.

Dahlander, L., and Magnusson, M. G. 2005. "Relationships between Open Source Software Companies and Communities: Observations from Nordic Firms," *Research Policy* (34), pp. 481-493.

Dahlander, L., and O'Mahony, S. 2011. "Progressing to the Center: Coordinating Project Work," *Organization Science* (22:4), pp. 961-979.
Dahlander, L., and Wallin, M. W. 2006. "A Man on the Inside: Unlocking Communities as Complementary Assets," *Research Policy* (35), pp. 1243-1259.
Daily, C. M., Dalton, D. R., and Rajagopalan, N. 2003. "Governance through Ownership: Centuries of Practice, Decades of Research," *Academy of Management Journal* (46:2), pp. 151-158.
Dalle, M., and David, P. A. 2005. "The Allocation of Software Development Resources in 'Open Source' Production Mode," in *Perspectives on Open Source and Free Software,* J. Feller, B. Fitzgerald, S. Hissam and K. Lakhani (eds.). Sebastapol, CA: O'Reilly Associates.
Davis, J. P., and Eisenhardt, K. M. 2011. "Rotating Leadership and Collaborative Innovation: Recombination Processes in Symbiotic Relationships," *Administrative Science Quarterly* (56:2), pp. 159-201.
de Laat, P. B. 2007. "Governance of Open Source Software: State of the Art," *Journal of Management and Governance* (11:2), pp. 165-177.
De Noni, I., Ganzaroli, A., and Orsi, L. 2011. "The Governance of Open Source Software Communities: An Exploratory Analysis," *Journal of Business Systems, Governance and Ethics* (6:1), pp. 1-18.
De Noni, I., Ganzaroli, A., and Orsi, L. 2013. "The Evolution of Oss Governance: A Dimensional Comparative Analysis," *Scandinavian Journal of Management* (29:3), pp. 247-263.
Demil, B., and Lecocq, X. 2006. "Neither Market nor Hierarchy nor Network: The Emergence of Bazaar Governance," *Organization Studies* (27:10), pp. 1447-1466.
Eisenhardt, K. M. 1985. "Control: Organizational and Economic Approaches " *Management Science* (31:2), pp. 134-149.
Eseryel, U. Y., and Eseryel, D. 2013. "Action-Embedded Transformational Leadership in Self-Managing Global Information Systems Development Teams," *The Journal of Strategic Information Systems* (22:2), pp. 103-120.
Etzioni, A. 1959. "Authority Structure and Organizational Effectiveness," *Administrative Science Quarterly* (4), pp. 43-67.
Faems, D., Janssens, M., Madhok, A., and Looy, B. V. 2008. "Toward an Integrative Perspective on Alliance Governance: Connecting Contract Design, Trust Dynamics, and Contract Application," *Academy of Management Journal* (51:6), pp. 1053-1078.
Farjoun, M. 2010. "Beyond Dualism: Stability and Change as a Duality," *The Academy of Management Review* (35:2), pp. 202-225.
Felin, T., and Zenger, T. R. 2014. "Closed or Open Innovation? Problem Solving and the Governance Choice," *Research Policy* (43:5), pp. 914-925.
Feller, J., Finnegan, P., Fitzgerald, B., and Hayes, J. 2008. "Bazaar by Design: Managing Interfirm Exchanges in an Open Source Service Network," *Information Technology in the Service Economy: Challenges and Possibilities for the 21st Century* (267), pp. 173-188.
Feller, J., and Fitzgerald, B. 2002. *Understanding Open Source Software Development*. London, UK: Addison-Wesley.
Fielding, R. T. 1999. "Shared Leadership in the Apache Project," *Communications of the ACM* (42:4).
Fitzgerald, B. 2006. "The Transformation of Open Source Software," *MIS Quarterly* (30:3), pp. 587-598.
Fogel, K. 1999. *Open Source Development with Cvs*. Scottsdale, AZ: Coriolis Open Press.
Fogel, K. 2005. *Producing Open Source Software: How to Run a Successful Free Software Project*. Sebastopol, CA: O'Reilly.
Gallivan, M. J. 2001. "Striking a Balance between Trust and Control in a Virtual Organization: A Content Analysis of Open Source Software Case Studies," *Information Systems Journal* (11), pp. 277-304.
Garud, R., Kumaraswamy, A., and Sambamurthy, V. 2006. "Emergent by Design: Performance and Transformation at Infosys Technologies," *Organization Science* (17:2), pp. 277-286.
Germonprez, M., Kendall, J. E., Kendall, K. E., Mathiassen, L., Young, B., and Warner, B. forthcoming. "A Theory of Responsive Design: A Field Study of Corporate Engagement with Open Source Communities," *Information Systems Research*).
Gerring, J. 2007. *Case Study Research: Principles and Practices*. Cambridge: Cambridge University Press.
Glaser, B. 1992. *Basics of Grounded Theory Analysis*. Mill Valley, California: Sociology Press.
Glaser, B. G., and Strauss, A. 1967. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Chicago: Aldine.
Gulati, R. 1998. "Alliances and Networks," *Strategic Management Journal* (19:4), pp. 293-317.

Gulati, R., and Singh, H. 1998. "The Architecture of Cooperation: Managing Coordination Costs and Appropriation Concerns in Strategic Alliances," *Administrative science quarterly* (43), pp. 781-814.
Han, K., Oh, W., Im, K. S., Chang, R. M., Oh, H., and Pinsonneault, A. 2012. "Value Cocreation and Wealth Spillover in Open Innovation Alliances," *MIS Quarterly* (36:1), pp. 291-325.
Hargadon, A. B., and Douglas, Y. 2001. "When Innovations Meet Institutions: Edison and the Design of the Electric Light," *Administrative Science Quarterly* (46:3), pp. 476-501.
Harrison, P. M. 1960. "Weber's Categories of Authority and Voluntary Associations," *American Sociological Review* (25:2), pp. 232-237.
Harrison, S., and Rouse, E. 2014. "Let's Dance! Elastic Coordination in Creative Group Work: A Qualitative Study of Modern Dancers," *Academy of Management Journal* (57:5), pp. 1256-1283.
Hemetsberger, A., and Reinhardt, C. 2009. "Collective Development in Open-Source Communities: An Activity Theoretical Perspective on Successful Online Collaboration," *Organization Studies* (30:9), pp. 987-1008.
Howison, J., and Crowston, K. 2014. "Collaboration through Open Superposition: A Theory of the Open Source Way," *MIS Quarterly* (38:1), pp. 29-50.
Ingram, P., and Torfason, M. T. 2010. "Organizing the in-Between: The Population Dynamics of Network-Weaving Organizations in the Global Interstate Network," *Administrative Science Quarterly* (55:4), pp. 577-605.
Jarzabkowski, P., Le, J. K., and Feldman, M. 2012. "Toward a Theory of Coordinating: Creating Coordinating Mechanisms in Practice," *Organization Science* (23:4), pp. 907-927.
Kallinikos, J. 2004. "The Social Foundations of the Bureaucratic Order," *Organization* (11:1), pp. 13-36.
Kellogg, K. C., Orlikowski, W. J., and Yates, J. 2006. "Life in the Trading Zone: Structuring Coordination across Boundaries in Postbureaucratic Organizations," *Organization Science* (17:1), pp. 22-44.
Kieser, A. 1994. "Why Organization Theory Needs Historical Analyses-and How This Should Be Performed," *Organization Science* (5:4), pp. 608-620.
Koch, S., and Schneider, G. 2002. "Effort, Cooperation and Coordination in an Open Source Software Project: Gnome," *Information Systems Journal* (12:1), pp. 27-42.
Kogut, B., and Metiu, A. 2001. "Open-Source Software Development and Distributed Innovation," *Oxford Review of Economic Policy* (17:2), pp. 248-264.
Kotlarsky, J., Scarbrough, H., and Oshri, I. 2014. "Coordinating Expertise across Knowledge Boundaries in Offshore-Outsourcing Projects: The Role of Codification," *MIS Quarterly* (38:2), pp. 607-627.
Krishnamurthy, S. 2002. "Cave or Community? An Empirical Examination of 100 Mature Open Source Projects," *First Monday* (http://www.firstmonday.dk/issues/issue7_6/krishnamurthy/index.html).
Kuwabara, K. 2000. "Linux: A Bazaar at the Edge of Chaos," *First Monday* (5:3).
Langley, A. 1999. "Strategies for Theorizing from Process Data," *Academy of Management review* (24), pp. 691-710.
Langley, A., Smallman, C., Tsoukas, H., and Van de Ven, A. H. 2013. "Process Studies of Change in Organization and Management: Unveiling Temporality, Activity, and Flow," *Academy of Management Journal* (56:1), pp. 1-13.
Lanzara, G. F., and Morner, M. 2003. "The Knowledge Ecology of Open Source Software Projects," *European Group of Organizational Studies (EGOS Colloquium)*, Copenhagen.
Lanzara, G. F., and Morner, M. 2005. "Artifacts Rule! How Organizing Happens in Open Source Projects," in *Actor-Network Theory and Organizing,* B. Czarniawska and T. Hernes (eds.). Copenhagen, Denmark: Copenhagen Business School Press.
Lee, G. K., and Cole, R. E. 2003. "From a Firm-Based to a Community-Based Model of Knowledge Creation: The Case of the Linux Kernel Development," *Organization Science* (14:6), pp. 633-649.
Lindberg, A., Berente, N., Gaskin, J., and Lyytinen, K. 2016. "Coordinating Interdependencies in Online Communities: A Study of an Open Source Software Project," *Information Systems Research* (27:4), pp. 751-772.
Machek, P. 2003 - Wed 26th Feb. "Bitbucket: Gpl-Ed Bitkeeper Clone." 2004, from http://www.uwsg.indiana.edu/hypermail/linux/kernel/0302.3/0931.html
Malone, T. W. 1987. "Modeling Coordination in Organizations and Markets," *Management Science* (33:10 (October)), pp. 1317-1332.

Markus, M. L. 2007. "The Governance of Free/Open Source Software Projects: Monolithic, Multidimensional, or Configurational?," *Journal of Management and Governance* (11:2), pp. 151-163.
Markus, M. L., and Bui, Q. N. 2012. "Going Concerns: The Governance of Interorganizational Coordination Hubs," *Journal of Management Information Systems* (28:4), pp. 163-198.
Mason, R. O., McKenney, J. L., and Copeland, D. G. 1997a. "Developing an Historical Tradition in Mis Research," *MIS Quarterly* (21:3), pp. 257-276.
Mason, R. O., McKenney, J. L., and Copeland, D. G. 1997b. "An Historical Method for Mis Research: Steps and Assumptions," *MIS Quarterly* (21:3), pp. 307-320.
McVoy, L. 2003 - Sat 1st March. "Re: Bitbucket: Gpl-Ed Bitkeeper Clone." 2004, from http://www.uwsg.indiana.edu/hypermail/linux/kernel/0303.0/0052.html
Meyer, M., and Montagne, F. 2007. "Open Source Software and the Self-Governed Community," *Revue D Economie Politique* (117:3), pp. 387-405.
Meyer, M. W. 1968. "The Two Authority Structures of Bureaucratic Organizatino," *Administrative Science Quarterly* (13:2), pp. 211-228.
Midha, V., and Bhattacherjee, A. 2012. "Governance Practices and Software Maintenance: A Study of Open Source Projects," *Decision Support Systems* (54:1), pp. 23-32.
Miller, D. 1987. "The Genesis of Configuration," *The Academy of Management Review* (12:4), pp. 686-701.
Miller, D. 1990. "Organizational Configurations: Cohesion, Change, and Prediction," *Human Relations* (43:8), pp. 771-789.
Mintzberg, H. 1980. "Structure in 5's: A Synthesis of the Research on Organization Design," *Management Science* (26:3), pp. 322-341.
Mintzberg, H., and McHugh, A. 1985. "Strategy Formation in an Adhocracy," *Administrative Science Quarterly* (30:2), pp. 160-197.
Mockus, A., Fielding, R. T., and Herbsleb, J. 2002. "Two Case Studies of Open Source Software Development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology (TOSEM)* (11:3), pp. 309 - 346.
Molnar, I. 2002 - Sun 6th Oct. "Bk Metadata License Problem?", 2004, from http://www.uwsg.indiana.edu/hypermail/linux/kernel/0210.0/1918.html
Moon, J. Y., and Sproull, L. 2000. "Essence of Distributed Work: The Case of the Linux Kernel," *First Monday* (5:11).
O'Mahony, S. 2007. "The Governance of Open Source Initiatives: What Does It Mean to Be Community Managed?," *Journal of Management & Governance* (11:2), pp. 139-150.
O'Mahony, S., and Ferraro, F. 2007. "The Emergence of Governance in an Open Source Community," *Academy of Management Journal* (50), pp. 1079-1106.
Oh, W., Moon, J. Y., Hahn, J., and Kim, T. 2016. "Research Note—Leader Influence on Sustained Participation in Online Collaborative Work Communities: A Simulation-Based Approach," *Information Systems Research* (27:2), pp. 383-402.
Olson, G. M., Malone, T. W., and Smith, J. B. 2001. *Coordination Theory and Collaboration Technology*. Psychology Press.
Olson, G. M., and Olson, J. S. 2000. "Distance Matters," *Human-computer interaction* (15:2), pp. 139-178.
Orek, S., and Nov, O. 2008. "Exploring Motivations for Contributing to Open Source Initiatives: The Roles of Contribution Context and Personal Values," *Computers in Human Behavior* (24), pp. 2055-2073.
Osborn, R. N., and Hagedoorn, J. 1997. "The Institutionalization and Evolutionary Dynamics of Interorganizational Alliances and Networks," *Academy of Management Journal* (40:2), pp. 261-278.
Ouchi, W. 1980. "Markets, Bureaucracies and Clans," *Administrative Science Quarterly* (25), pp. 120-142.
Peng, G., Wan, Y., and Woodlock, P. 2013. "Network Ties and the Success of Open Source Software Development," *The Journal of Strategic Information Systems* (22:4), pp. 269-281.
Pentland, B. T. 1999. "Building Process Theory with Narrative: From Description to Explanation," *Academy of Management Review* (24:4), pp. 711-724.
Raymond, E. 1999. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. Sebastopol, California: O'Reilly & Associates.
Ribes, D., Jackson, S., Geiger, S., Burton, M., and Finholt, T. 2013. "Artifacts That Organize: Delegation in the Distributed Organization," *Information and Organization* (23:1), pp. 1-14.

Riccucci, N. M., Van Ryzin, G. G., and Li, H. 2016. "Representative Bureaucracy and the Willingness to Coproduce: An Experimental Study," *Public Administration Review* (76:1), pp. 121-130.
Ring, P. S., and van de Ven, A. H. 1992. "Structuring Cooperative Relationships between Organizations," *Strategic Management Journal* (13:7), pp. 483-498.
Roberts, J., Hann, I.-H., and Slaughter, S. 2006. "Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects," *Management Science* (52:7), pp. 984-999.
Scarbrough, H., Panourgias, N. S., and Nandhakumar, J. 2014. "Developing a Relational View of the Organizing Role of Objects: A Study of the Innovation Process in Computer Games," *Organization Studies*).
Schmidt, K., and Bannon, L. 1992. "Taking Cscw Seriously," *Computer supported cooperative work (CSCW)* (1:1), pp. 7-40.
Schmidt, K., and Simonee, C. 1996. "Coordination Mechanisms: Towards a Conceptual Foundation of Cscw Systems Design," *Computer supported cooperative work (CSCW)* (5:2), pp. 155-200.
Shah, S. K. 2006. "Motivation, Governance, and the Viability of Hybrid Forms in Open Source Software Development," *Management Science* (52:7), pp. 1000-1014.
Shaikh, M., and Vaast, E. 2016. "Folding and Unfolding: Balancing Openness and Transparency in Open Source Communities," *Information Systems Research* (27:4), pp. 813-833.
Sharma, S., Sugumaran, V., and Rajagopalan, B. 2002. "A Framework for Creating Hybrid-Open Source Software Communities," *Information Systems Journal* (12:1), pp. 7-26.
Spaeth, S., von Krogh, G., and He, F. 2015. "Research Note—Perceived Firm Attributes and Intrinsic Motivation in Sponsored Open Source Software Projects," *Information Systems Research* (26:1), pp. 224-237.
Stanko, M. A. 2016. "Toward a Theory of Remixing in Online Innovation Communities," *Information Systems Research* (27:4), pp. 773-791.
Strauss, A. 1987. *Qualitative Analysis for Social Scientists*. Cambridge: Cambridge University Press.
Strauss, A., and Corbin, J. 1998. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. Thousand Oaks, CA: Sage Publications.
Strode, D. E., Huff, S. L., Hope, B., and Link, S. 2012. "Coordination in Co-Located Agile Software Development Projects," *Journal of Systems and Software* (85:6), pp. 1222-1238.
Sundararajan, A., Provost, F., Oestreicher-Singer, G., and Aral, S. 2013. "Research Commentary—Information in Digital, Economic, and Social Networks," *Information Systems Research* (24:4), pp. 883-905.
Tsoukas, H. 1989. "The Validity of Idiographic Research Explanations," *Academy of Management Review* (14:4), pp. 551-561.
Tullio, D. D., and Staples, D. S. 2014. "The Governance and Control of Open Source Software Projects " *Journal of Management Information Systems* (30:3), pp. 49-80
Venters, W., Oborn, E., and Barrett, M. 2014. "A Trichordal Temporal Approach to Digital Coordination: The Sociomaterial Mangling of the Cern Grid," *MIS Quarterly* (38:3), pp. 927-A918.
von Hippel, E. 2001. "Innovation by User Communities: Learning from Open-Source Software," *MIT Sloan Management Review* (42:4), pp. 82-86.
von Krogh, G., Haefliger, S., Spaeth, S., and Wallin, W. 2012. "Carrots and Rainbows: Motivation and Social Practice in Open Source Software Development," *MIS Quarterly* (36:2), pp. 649-676.
von Krogh, G., Spaeth, S., and Haefliger, S. 2005. "Knowledge Reuse in Open Source Software: An Exploratory Study of 15 Open Source Projects," *HICSS*, Hawaii.
von Krogh, G., and von Hippel, E. 2006. "The Promise of Research on Open Source Software," *Management Science* (52:7), pp. 975-983.
Weber, M. 1946. *Bureaucracy. From Max Weber: Essays in Sociology*. New York: Oxford University Press
Weber, S. 2005. "Patterns of Governance in Open Source," in *Open Sources 2.0: The Continuing Evolution,* C. DiBona, M. Stone and D. Cooper (eds.). Sebastopol, CA: O'Reilly, pp. 361-372.

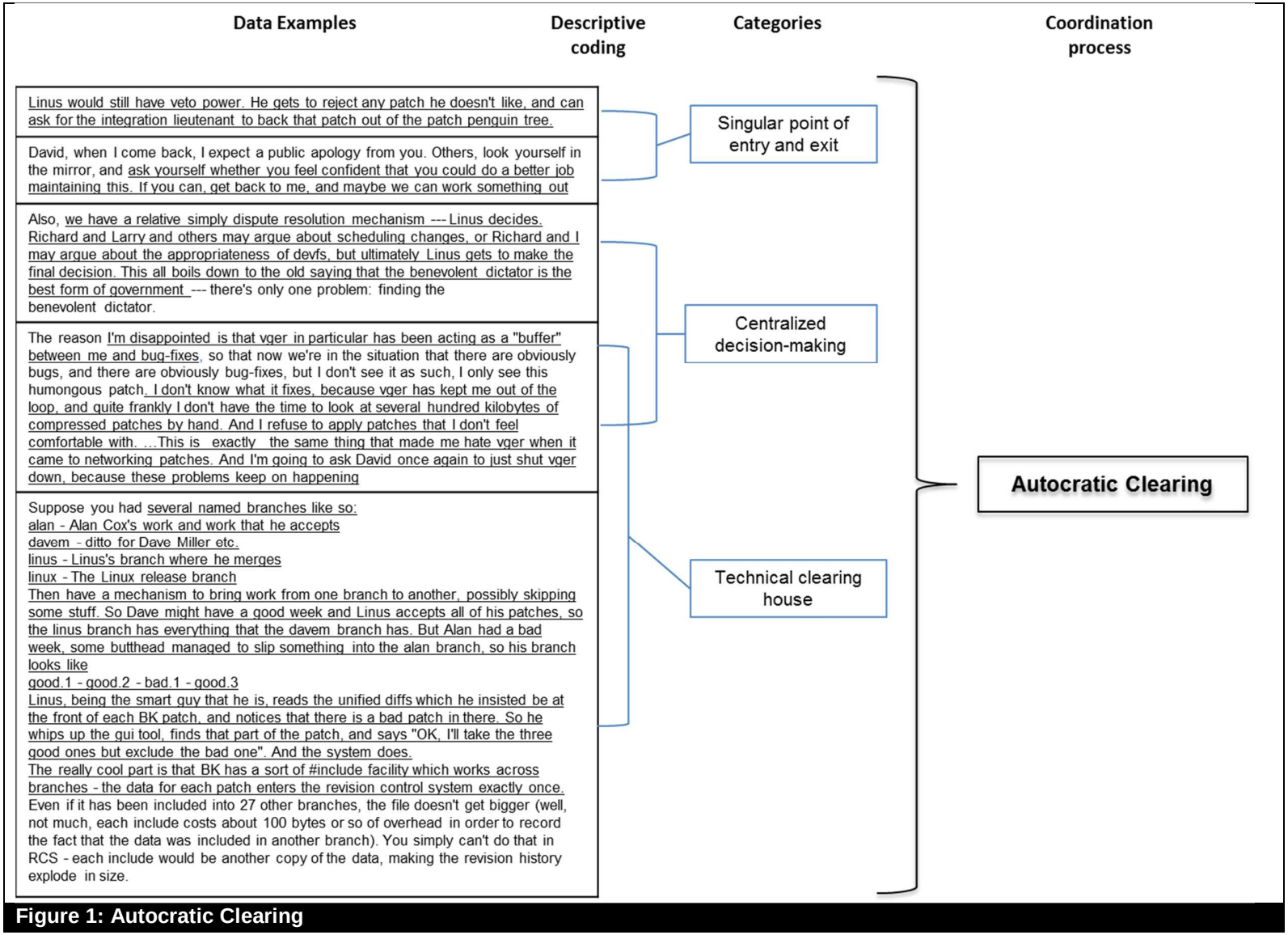

Figure 1: Autocratic Clearing

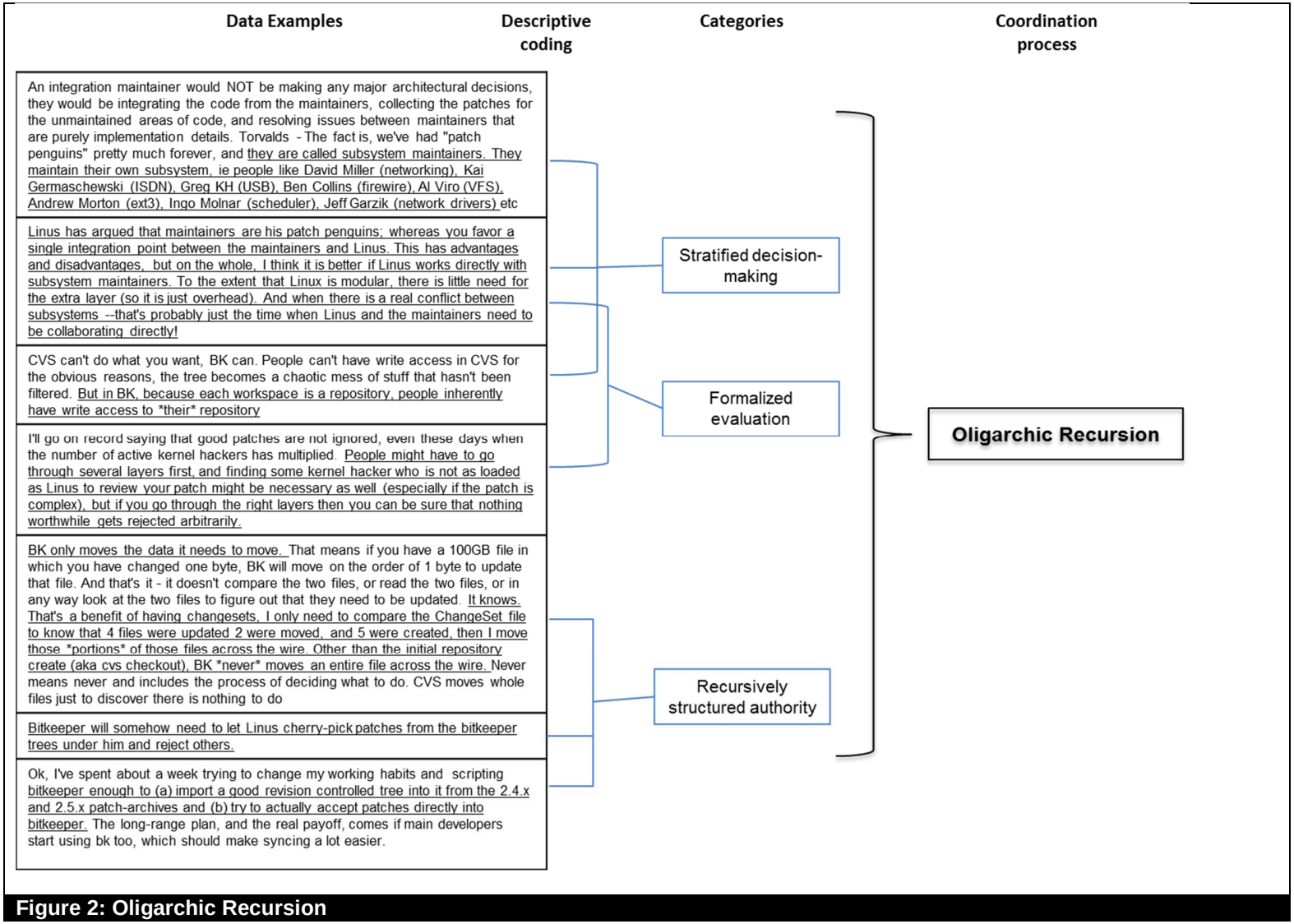

Figure 2: Oligarchic Recursion

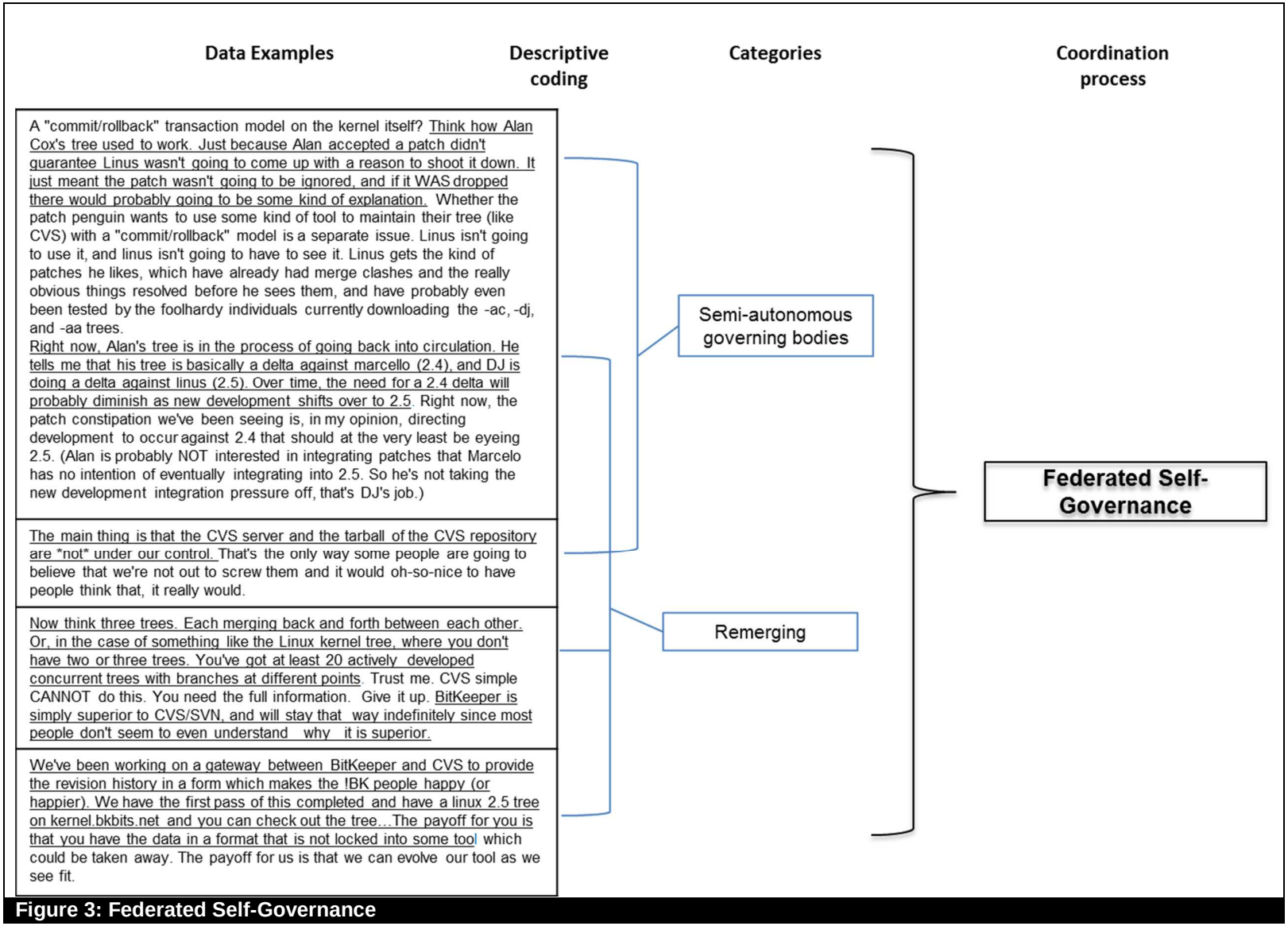

Figure 3: Federated Self-Governance

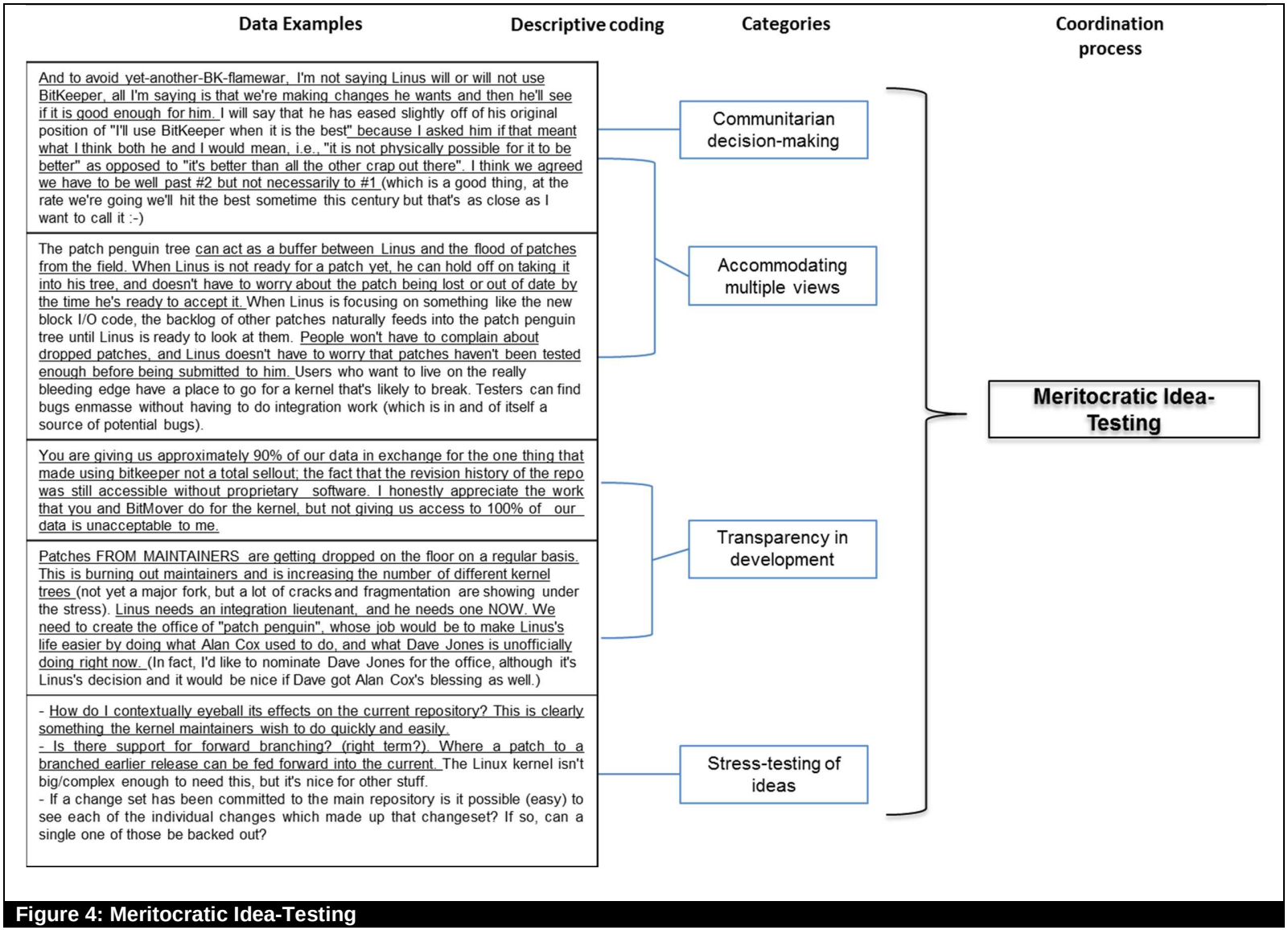

Figure 4: Meritocratic Idea-Testing